IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of	:	
Takuji KAWAMOTO	:	
Serial No. NEW	:	**Attn: APPLICATION BRANCH**
Filed March 18, 2004	:	Attorney Docket No. 2004-0445A

DATA MEMORY CACHE UNIT AND DATA MEMORY CACHE SYSTEM

CLAIM OF PRIORITY UNDER 35 USC 119

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Applicant in the above-entitled application hereby claims the date of priority under the International Convention of Japanese Patent Application No. 2003-078026, filed March 20, 2003, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Takuji KAWAMOTO

By _____
Jeffrey R. Filipek
Registration No. 41,471
Attorney for Applicant

JRF/krg
Washington, D.C. 20006-1021
Telephone (202) 721-8200
Facsimile (202) 721-8250
March 18, 2004

THE COMMISSIONER IS AUTHORIZED TO CHARGE ANY DEFICIENCY IN THE FEES FOR THIS PAPER TO DEPOSIT ACCOUNT NO. 23-0975

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００３年　３月２０日

出願番号
Application Number: 特願２００３－０７８０２６

[ST. 10/C]: ［ＪＰ２００３－０７８０２６］

出願人
Applicant(s): 松下電器産業株式会社

２００３年１０月２９日

特許庁長官
Commissioner,
Japan Patent Office 今井康夫

出証番号　出証特２００３－３０８９５９５

【書類名】　特許願
【整理番号】　2968150011
【提出日】　平成15年 3月20日
【あて先】　特許庁長官殿
【国際特許分類】　G06F 12/08
【発明者】
　【住所又は居所】　愛知県名古屋市中区栄２丁目６番１号白川ビル別館５階
　株式会社松下電器情報システム名古屋研究所内
　【氏名】　川本　琢二
【特許出願人】
　【識別番号】　000005821
　【氏名又は名称】　松下電器産業株式会社
【代理人】
　【識別番号】　100097445
　【弁理士】
　【氏名又は名称】　岩橋　文雄
【選任した代理人】
　【識別番号】　100103355
　【弁理士】
　【氏名又は名称】　坂口　智康
【選任した代理人】
　【識別番号】　100109667
　【弁理士】
　【氏名又は名称】　内藤　浩樹
【手数料の表示】
　【予納台帳番号】　011305
　【納付金額】　21,000円
【提出物件の目録】
　【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9809938

【書類名】　明細書

【発明の名称】　データメモリキャッシュ制御装置及びデータメモリキャッシュ制御方法

【特許請求の範囲】

【請求項１】　ＣＰＵがアクセスするメモリ領域のアクセスするアドレスの連続性に関する情報とアクセスするアドレスの方向性に関する情報に基づいてキャッシュメモリの制御を行う、

データメモリキャッシュ制御装置。

【請求項２】　ＣＰＵからの連続したアドレスのメモリ領域に対する連続書き込み時にキャッシュミスが発生した時、次に書き込みが行われるアドレス方向のメモリ領域に対しては主記憶装置からキャッシュメモリへの読み込みを行わない、データキャッシュメモリ制御装置。

【請求項３】　ＣＰＵによる連続したアドレスのメモリ領域に対する連続読み込み時にキャッシュミスが発生した時、既に読み込みが行われたアドレス方向のメモリ領域に対してはキャッシュメモリから主記憶装置への書き込みを行わない、データキャッシュメモリ制御装置。

【請求項４】　ＣＰＵからの連続したアドレスのメモリ領域に対する連続書き込み時に、既に書き込みが行われたアドレス方向のメモリ領域で、現在書き込みが行われているメモリ領域とのアドレス距離が一定以上離れたメモリ領域に対して、バスの空き時間を使って、予めキャッシュメモリから主記憶装置への書き込みを行う、データキャッシュメモリ制御装置。

【請求項５】　ＣＰＵによる連続したアドレスのメモリ領域に対する連続読み込み時に、今後読み込みが行われるアドレス方向のメモリ領域で、現在読み込みが行われているメモリ領域とのアドレス距離が一定以上近いメモリ領域に対して、バスの空き時間を使って、予め主記憶装置からキャッシュメモリへの読み込みを行う、データキャッシュメモリ制御装置。

【請求項６】　ＣＰＵがアクセスするメモリ領域のアクセスするアドレスの連続性に関する情報とアクセスするアドレスの方向性に関する情報に基づいてキャッシュメモリの制御を行う、

データメモリキャッシュ制御方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本特許出願に係る発明（以後単に「本発明」とも言う）は、ＣＰＵが主記憶装置をアクセスする時、主記憶装置のアクセス速度が遅いことによるオーバーヘッドを回避し、パフォーマンスを向上させることを目的とするメモリキャッシュ（キャッシュメモリとも言う）の制御装置及び制御方法に関するものである。

【０００２】

特に、データを記憶するデータメモリのキャッシュに関するものであり、更にいわゆるスタックメモリ、即ち、プッシュ・ポップ命令によってアドレスが連続的にアクセス（データの読み込みや書き込み）が行われるデータメモリのキャッシュに関するものである。

【０００３】

【従来の技術】

従来の、プッシュ・ポップ命令に必要な時間を減少させ、キャッシュ内にプッシュ・ポップ後の不要なデータを保持しないようにするための、スタックキャッシュの制御方式及びスタックキャッシュには、例えば、主記憶装置上のデータ構造として最後に入力したデータが最初に出力されるスタック構造をサポートし、スタックへのデータ入力をプッシュ、前記スタックからのデータ出力をポップとして、他のデータ書込み／読出しと区別してデータ処理を行うマイクロコンピュータを備えたマイクロコンピュータシステムにおいて、前記マイクロコンピュータがプッシュするとき前記主記憶装置に代わって該当アドレスとデータの組を保持し、前記主記憶装置にプッシュされていない保持データを前記マイクロコンピュータが外部にアクセスしていない時を見計らって上記主記憶装置にプッシュし、前記マイクロコンピュータがポップするとき該当データを保持していれば前記主記憶装置に代わって該当データを前記マイクロコンピュータに出力し、前記マイクロコンピュータのポップで不要になったデータを後にポップされる可能性のあるスタック底部のデータに置き換えるため、前記マイクロコンピュータが外部

にアクセスしていない時を見計らって前記主記憶装置からポップすることを特徴とするスタックキャッシュの制御方式があった（例えば、特許文献１参照。）。

【０００４】

【特許文献１】

特開平５－１４３３３０号公報

【０００５】

【発明が解決しようとする課題】

しかしながら前記従来のスタックキャッシュ制御装置及びスタックキャッシュ制御方法では、通常のデータメモリに対するランダムアクセスを前提としたメモリアクセス高速化手法と何等変わることなく、スタックメモリと言う規則性のあるアクセスの特徴を生かすことができず、高速化の効率に問題があった。

【０００６】

このことは特に近年ユビキタスコンピューティング等のためにニーズが増大しているＪａｖａ（Ｒ）（ＴＭ）システム等、スタックアクセスの比重が大きいメモリシステムでは特に大きな問題となる。

【０００７】

具体的な問題点を次に説明する。

【０００８】

例えばスタックに対するデータプッシュが、主記憶装置上のメモリアドレスの増加方向に行われたと仮定し、更にこのスタックがキャッシュメモリにキャッシュされていたと仮定する。

【０００９】

そしてあるデータＡのプッシュによってキャッシュミスが発生したと仮定する。

【００１０】

この条件下では、キャッシュミスが発生したメモリアドレスから１キャッシュ単位分の新しいデータメモリの内容が、主記憶装置からキャッシュメモリに読み込まれる。

【００１１】

この状態を図２５に示す。図２５はアドレス９０のメモリ領域にスタックポインタ９１３があり、このアドレス９０のメモリ領域にデータＡがプッシュされた状態を示している。

【００１２】

この時、図２５に示すようにデータＡのプッシュによってスタックポインタ９１３がアドレス８９のメモリ領域からアドレス９０のメモリ領域に移動する。

【００１３】

そして更に同じ図２５に示すようにアドレス９０、９１、９２、９３のメモリ領域が１つのキャッシュ単位であり、この１つのキャッシュ単位のアドレス９０のメモリ領域が始めてアクセスされたので、このアドレス９０、９１、９２、９３のメモリ領域からなる１キャッシュ単位が、主記憶装置からキャッシュメモリに読み込まれる。

【００１４】

しかしながらこのキャッシュメモリに読み込まれた１キャッシュ単位分の新しいデータメモリの内容は、スタックアクセスの規則性に従う限り、続けて行われるデータプッシュによって上書きされる内容であり、決して読み出されることは有り得ないデータである。

【００１５】

そしてこのように決して読み出されることの有り得ないデータを、主記憶装置からキャッシュメモリに読み込むことは高速化の効率に大きな弊害であった。

【００１６】

逆にスタックに対するデータポップが、主記憶装置上のメモリアドレスの減少方向に行われたと仮定し、更にこのスタックがキャッシュメモリにキャッシュされていたと仮定する。

【００１７】

そしてあるデータＢのポップによってキャッシュミスが発生したと仮定する。

【００１８】

この条件下では、キャッシュミスが発生したメモリアドレスよりも大きなメモリアドレスから１キャッシュ単位分のデータメモリの内容が、キャッシュメモリ

から主記憶装置にライトバックされる。

【００１９】

この状態を図２６に示す。図２６に示すようにデータＢのポップによってスタックポインタ９２３がアドレス９０のメモリ領域からアドレス８９のメモリ領域に移動する。

【００２０】

そして更に同じ図２６に示すようにアドレス９０、９１、９２、９３のメモリ領域が１つのキャッシュ単位であり、この１つのキャッシュ単位以外のアドレス８９のメモリ領域にスタックポインタが移動したので、このアドレス９０、９１、９２、９３のメモリ領域からなる１キャッシュ単位が、キャッシュメモリから主記憶装置にライトバックされ、キャッシュメモリが開放される。

【００２１】

しかしながらこの主記憶装置にライトバックされた１キャッシュ単位分のデータメモリの内容は、スタックアクセスの規則性に従う限り、既に行われたデータポップによって読み出された内容であり、決して再び読み出されることは有り得ない。そしてこのように決して読み出されることの有り得ないデータを、キャッシュメモリから主記憶装置にライトバックすることは高速化の効率に大きな弊害であった。

【００２２】

また、スタックに対するデータプッシュが、主記憶装置上のメモリアドレスの増加方向に行われたと仮定し、更にこのスタックがキャッシュメモリにキャッシュされていたと仮定する。

【００２３】

そしてあるデータＣ、Ｄ、・・、Ｅのプッシュが連続して実行されたと仮定する。

【００２４】

この条件下では、現在スタックに対してプッシュ・ポップを行う領域であるスタックポインタの示すアドレスよりも低いアドレスのメモリ領域に多くのスタックプッシュされたデータ、今の例ではＣやＤが記憶されており、この低いアドレ

スのメモリ領域に記憶されたデータは、暫くの時間的な間、ポップによって読み出される可能性は非常に低い。

【００２５】

この状態を図２７に示す。図２７はアドレス１９４のメモリ領域にスタックポインタ９３３があり、遥か下の例えばアドレス８０のメモリ領域からこのアドレス１９４のメモリ領域まで、連続してＣ、Ｄ、・・、Ｅのデータがプッシュされた状態を示している。

【００２６】

この時図２７に示すように、通常ならばアドレス８０からアドレス８３までの１つのキャッシュ単位はキャッシュメモリがフルにならない限りキャッシュメモリ領域を占めたままであり、主記憶装置にライトバックされることはない。

【００２７】

そしてキャッシュメモリがフルになった時に始めて、主記憶装置にライトバックされ、キャッシュメモリを他のメモリ領域のために明け渡すことになる。

【００２８】

このようにキャッシュメモリがゼロになり、新しいキャッシュメモリ領域に対する要求が発生してからライトバックを行うことは高速化の効率に大きな弊害であった。

【００２９】

例えば図２７に示すように、現在のスタックポインタ９３３の位置であるアドレス１９４のメモリ領域よりも遥かに下のメモリ領域、例えばアドレス８０～８３のメモリ領域は当分の間、ポップによって読み出される可能性は非常に低く、例えばこのようなアドレス８０～８３のメモリ領域である１つのキャッシュ単位のキャッシュメモリの内容を、予め主記憶装置にライトバックしておくことによって、キャッシュメモリの余裕がなくなった段階で慌ててキャッシュメモリの内容を主記憶装置に書き込み、キャッシュメモリに空き領域を作るよりも、メモリアクセス高速化の効率を大きく改善することができる。

【００３０】

更にまた、スタックに対するデータポップが、主記憶装置上のメモリアドレス

の減少方向に行われたと仮定し、更にこのスタックがキャッシュメモリにキャッシュされていたと仮定する。

【００３１】

そしてあるデータＦ、Ｇ、・・、Ｈのポップが連続して実行されたと仮定する。

【００３２】

この条件下では、現在スタックに対してプッシュ・ポップを行う領域であるスタックポインタの示すアドレスよりも低い値のアドレス領域、例えば今の例ではアドレス８４～８７のメモリ領域にスタックプッシュされているデータは、すぐに続けてポップによって読み出される可能性が非常に高い。

【００３３】

この状態を図２８に示す。図２８はアドレス９１のメモリ領域にスタックポインタ９４３があり、遥か上の例えばアドレス１９６のメモリ領域からこのアドレス９１のメモリ領域まで、連続してＦ、Ｇ、・・、Ｈのデータがポップされた状態を示している。

【００３４】

この時図２８に示すように、通常ならばアドレス８４からアドレス８７までの１つのキャッシュ単位はこの領域に属する１つのメモリ、例えばアドレス８７のメモリ領域がポップの実行によってアクセスされない限り主記憶装置からキャッシュメモリ領域に読み込まれることはない。

【００３５】

そして例えばこのアドレス８７のメモリ領域がポップの実行によってアクセスされた時に始めて、主記憶装置からキャッシュメモリ上に読み出されることになる。

【００３６】

このように新しい１つのキャッシュメモリ単位に属する領域が始めてアクセスされ、新しい１つのキャッシュメモリ領域に対するデータの要求が発生した後に、主記憶装置からキャッシュメモリ領域にデータの読み込みを行うことは高速化の効率に大きな弊害であった。

【００３７】

例えば図２８に示すように、現在のスタックポインタ９４３の位置であるアドレス９１のメモリ領域のすぐ下のメモリ領域、例えばアドレス８４～８７のメモリ領域を占める１つのキャッシュ単位は、すぐに続けてポップによって読み出される可能性が非常に高く、例えばこのようなアドレス８４～８７のメモリ領域である１つのキャッシュ単位のデータを、予め主記憶装置からキャッシュメモリに読み込んでおくことによって、キャッシュメモリ上にないメモリ領域に対するアクセスが行われた段階で慌てて主記憶装置の内容をキャッシュメモリに読み込むよりも、メモリアクセス高速化の効率を大きく改善することができる。

【００３８】

【課題を解決するための手段】

前記課題を解決するため、本特許出願に係る請求項１に記載の発明は、ＣＰＵがアクセスするメモリ領域のアクセスするアドレスの連続性に関する情報とアクセスするアドレスの方向性に関する情報に基づいてキャッシュメモリの制御を行う、データメモリキャッシュ制御装置である。

【００３９】

前記課題を解決するため、本特許出願に係る請求項２に記載の発明は、ＣＰＵからの連続したアドレスのメモリ領域に対する連続書き込み時にキャッシュミスが発生した時、次に書き込みが行われるアドレス方向のメモリ領域に対しては主記憶装置からキャッシュメモリへの読み込みを行わない、データキャッシュメモリ制御装置である。

【００４０】

前記課題を解決するため、本特許出願に係る請求項３に記載の発明は、ＣＰＵによる連続したアドレスのメモリ領域に対する連続読み込み時にキャッシュミスが発生した時、既に読み込みが行われたアドレス方向のメモリ領域に対してはキャッシュメモリから主記憶装置への書き込みを行わない、データキャッシュメモリ制御装置である。

【００４１】

前記課題を解決するため、本特許出願に係る請求項４に記載の発明は、ＣＰＵ

からの連続したアドレスのメモリ領域に対する連続書き込み時に、既に書き込みが行われたアドレス方向のメモリ領域で、現在書き込みが行われているメモリ領域とのアドレス距離が一定以上離れたメモリ領域に対して、バスの空き時間を使って、予めキャッシュメモリから主記憶装置への書き込みを行う、データキャッシュメモリ制御装置である。

【００４２】

前記課題を解決するため、本特許出願に係る請求項５に記載の発明は、ＣＰＵによる連続したアドレスのメモリ領域に対する連続読み込み時に、今後読み込みが行われるアドレス方向のメモリ領域で、現在読み込みが行われているメモリ領域とのアドレス距離が一定以上近いメモリ領域に対して、バスの空き時間を使って、予め主記憶装置からキャッシュメモリへの読み込みを行う、データキャッシュメモリ制御装置である。

【００４３】

前記課題を解決するため、本特許出願に係る請求項６に記載の発明は、ＣＰＵがアクセスするメモリ領域のアクセスするアドレスの連続性に関する情報とアクセスするアドレスの方向性に関する情報に基づいてキャッシュメモリの制御を行う、データメモリキャッシュ制御方法である。

【００４４】

【発明の実施の形態】

本特許出願に係る発明（以後、「本発明」とも言う）の実施の形態について図を参照して説明する。

【００４５】

（実施の形態１）

本発明の第１の実施の形態であるデータメモリキャッシュ制御装置（以後、特に区別するべき場合を除いて単に「キャッシュ制御装置」と言う）のブロック構成を図１に示す。命令メモリキャッシュ制御装置については本発明と直接関係しないので説明を省略する。

【００４６】

本実施の形態ではキャッシュ制御装置は大きく、演算ユニット１１３、メモリ

マネジメントユニット１１５（以後、「MMU１１５」と言うこともある）、スタック対応データキャッシュユニット１１７（以後特に区別するべき場合を除いて単に「キャッシュユニット１１７」と言うこともある）、転送制御ユニット１１９、主記憶装置１３１から構成される。

【００４７】

演算ユニット１１３は通常ＭＰＵまたは（狭義の）ＣＰＵまたはプロセッサと呼ばれるもので各種の演算命令、データ移動命令、制御命令その他の命令を命令メモリから取り出して実行する。

【００４８】

以後、これらの一例としてＭＰＵを考え、「ＭＰＵ１１３」と言うが、これらを全て含むものを示している。

【００４９】

ＭＰＵ１１３は特にメモリアドレスを記憶する専用レジスタＳＰ（スタックポインタ）を有している。このスタックポインタが示すメモリ番地をスタックトップと呼ぶ。

【００５０】

またこのスタックポインタとして、専用のＳＰ以外に、汎用のレジスタＲｍ（レジスタｍ）を使うこともできる。このような実装は汎用ＣＰＵをＪａｖａ（Ｒ）仮想マシンとして使用する場合等で、特に有効である。

【００５１】

以後特に混乱が起こらない限り、「スタックポインタ（ＳＰ）」または「スタックポインタ（ＳＰ）アクセス」等の用語を、専用のＳＰレジスタを使う場合と、汎用のレジスタ（例えばＲｍやＲｎと表記する）を使用する場合を特に区別せず、両方を含む意味で使って説明する。

【００５２】

演算ユニットはデータ移動命令の一部として、ＳＰの内容をマイナス４してそのスタックトップにデータを４バイトライト（ＭＰＵ１１３からメモリへ書き込み）するｐｕｓｈ命令と、スタックトップからデータを例えば４バイトリード（メモリからＭＰＵ１１３へ読み出し）してＳＰの内容をプラス４するｐｏｐ命令

と、スタックトップからの相対アドレスでメモリ読み出しを行うｌｏａｄ命令と、スタックトップからの相対アドレスでメモリ書き込みを行うｓｔｏｒｅ命令と、その他直接アドレス、レジスタ間接アドレス、レジスタ相対アドレス等、各種の方法で指定する任意のアドレスに直接ランダムにメモリアクセスする命令と、その他の命令を備えている。

【００５３】

図２（ａ）、（ｂ）、（ｃ）、（ｄ）の左側「命令」欄に、このｐｕｓｈ、ｐｏｐ、ｌｏａｄ、ｓｔｏｒｅ命令の各命令形式を示す。これらは汎用レジスタＲｍまたは汎用レジスタＲｎをスタックポインタとして使用している例である。

【００５４】

また、この説明では、これらの命令によるメモリアクセスの単位を４バイトとし、ｐｕｓｈ／ｐｏｐ命令でデータ移動の前後でスタックポインタの増減を行う単位値を４としたが、これは１例であって、どのような値であっても構わない。

【００５５】

図２（ａ）左側の「命令」欄はｐｕｓｈ命令の命令形式を表している。この形式ではＲｍレジスタをスタックポインタとして使用している。図２（ａ）右側の「動作」欄にはこの命令の実行によって行われる動作が示されている。

【００５６】

このｐｕｓｈ命令が実行されるとまず、Ｒｍの内容がマイナス４される。

【００５７】

次にＲｎレジスタの内容がＲｍ間接アドレス指定でメモリに書き込まれる。即ち、Ｒｎレジスタの内容がＲｍレジスタの内容が示すアドレスのメモリに書き込まれる。

【００５８】

そしてｐｕｓｈ／ｐｏｐ修飾ビットがオンされる。

【００５９】

このｐｕｓｈ／ｐｏｐ修飾ビットの内容は、図１に示すように、制御ラインを通じて、ＭＰＵ１１３からキャッシュユニット１１７にｐｕｓｈ／ｐｏｐ命令が実行されたことを知らせるために使われる。

【００６０】

図２（ｂ）左側の「命令」欄はｐｏｐ命令の命令形式を表している。この形式ではＲｎレジスタをスタックポインタとして使用している。図２（ｂ）右側の「動作」欄にはこの命令の実行によって行われる動作が示されている。

【００６１】

このｐｏｐ命令が実行されるとまず、ｐｕｓｈ／ｐｏｐ修飾ビットがオンされる。

【００６２】

ｐｕｓｈ／ｐｏｐ修飾ビットの内容が、制御ラインを通じて、ＭＰＵ１１３からキャッシュユニット１１７にｐｕｓｈ／ｐｏｐ命令が実行されたことを知らせるために使われることは前記と同様である。

【００６３】

次に、Ｒｎ間接アドレス指定でメモリの内容がＲｍレジスタに読み込まれる。即ち、Ｒｎレジスタの内容が示すアドレスのメモリの内容がＲｍレジスタに読み込まれる。

【００６４】

そして、Ｒｎの内容がプラス４される。

【００６５】

図２（ｃ）左側の「命令」欄はｌｏａｄ命令の命令形式を表している。この形式ではＲｎレジスタをスタックポインタとして使用している。図２（ｃ）右側の「動作」欄にはこの命令の実行によって行われる動作が示されている。

【００６６】

このｌｏａｄ命令が実行されると、まずＳＰ相対修飾ビットがオンされる。

【００６７】

このＳＰ相対修飾ビットの内容は、図１に示すように、制御ラインを通じて、ＭＰＵ１１３からキャッシュユニット１１７にＳＰ相対アドレス指定でｌｏａｄ／ｓｔｏｒｅｐｏｐ命令が実行されたことを知らせるために使われる。

【００６８】

次に、Ｒｎレジスタの内容にオフセット値を加えた間接アドレス指定でメモリ

の内容がＲｍレジスタに読み込まれる。即ち、Ｒｎレジスタの内容にオフセット値を加えた値が示すアドレスのメモリの内容がＲｍレジスタに読み込まれる。

【００６９】

ＳＰとして使用されているＲｎの内容がプラスマイナスされることはない。

【００７０】

図２（ｄ）左側の「命令」欄はｓｔｏｒｅ命令の命令形式を表している。この形式ではＲｍレジスタをスタックポインタとして使用している。図２（ｄ）右側の「動作」欄にはこの命令の実行によって行われる動作が示されている。

【００７１】

このｓｔｏｒｅ命令が実行されると、Ｒｎレジスタの内容がＲｍレジスタの内容にオフセット値を加えた間接アドレス指定でメモリに書き込まれる。即ち、Ｒｎレジスタの内容がＲｍレジスタの内容にオフセット値を加えた値が示すアドレスのメモリに書き込まれる。

【００７２】

そしてＳＰ相対修飾ビットがオンされる。

【００７３】

ＳＰ相対修飾ビットの内容が、制御ラインを通じて、ＭＰＵ１１３からキャッシュユニット１１７にＳＰ相対アドレス指定でｌｏａｄ／ｓｔｏｒｅｐｏｐ命令が実行されたことを知らせるために使われることは前記と同様である。

【００７４】

ＳＰとして使用されているＲｎの内容がプラスマイナスされることはないことも前記と同様である。

【００７５】

これらの命令を含むデータ移動命令がＭＰＵ１１３によって実行されると、そのデータ移動メモリが外部メモリにアクセスするものであるならば、ＭＰＵ１１３はアクセス対象である外部メモリの論理アドレスを仮想アドレスバスに出力し、データアクセスがデータの読み出しであるか書き込みであるかに応じてキャッシュリード要求またはキャッシュライト要求ビット信号線をそれぞれオンにする。

【００７６】

データアクセスがデータのメモリへの書き込みであるならば同時に書き込み対象となるデータを３２ビット幅のデータバスにも出力する。

【００７７】

仮想データバスに出力された論理アドレスはＭＭＵ１１５によって物理アドレスに変換され、実アドレスバスに出力される。

【００７８】

これらのデータを受け取ったキャッシュユニット１１７は、データアクセスの対象であるメモリがキャッシュ上に存在するか否かを判定する。

【００７９】

以上説明したように本実施の形態で使用するＭＰＵからメモリアクセスするために使用する制御信号線は、ｐｕｓｈ／ｐｏｐ修飾、ＳＰ相対修飾、キャッシュリード要求、キャッシュライト要求の4本（４ビット）である。ＭＰＵによって実行される命令とその動作と各制御線の状態との取り得る組み合わせを図３に示す。

【００８０】

図３（ａ）はキャッシュリード要求とキャッシュライト要求がどちらもがオフ（０）であり、ｐｕｓｈ／ｐｏｐ修飾、ＳＰ相対修飾の如何に関わらず、キャッシュメモリのリードライトに関わる何等の動作も行われない。

【００８１】

図３（ｂ）はキャッシュリード要求がオフ（０）、キャッシュライト要求がオン（１）、ｐｕｓｈ／ｐｏｐ修飾とＳＰ相対修飾のどちらもがオフ（０）であり、メモリに対するランダムライトの命令が実行された時の動作とその時の制御信号線の状態が示される。

【００８２】

図３（ｃ）はキャッシュリード要求がオフ（０）、キャッシュライト要求がオン（１）、ｐｕｓｈ／ｐｏｐ修飾がオフ（０）、ＳＰ相対修飾がオン（１）であり、メモリに対するＳＰ相対ライト（ｓｔｏｒｅ）命令が実行された時の動作とその時の制御信号線の状態が示される。

【００８３】

図３（ｄ）はキャッシュリード要求がオフ（０）、キャッシュライト要求がオン（１）、ｐｕｓｈ／ｐｏｐ修飾がオン（１）、ＳＰ相対修飾がオフ（０）であり、メモリに対するｐｕｓｈ命令が実行された時の動作とその時の制御信号線の状態が示される。

【００８４】

図３（ｅ）はキャッシュリード要求がオフ（０）、キャッシュライト要求がオン（１）、ｐｕｓｈ／ｐｏｐ修飾とＳＰ相対修飾のどちらもがオン（１）であり、通常このような状態は想定されておらず、このような状態は起らない。

【００８５】

図３（ｆ）はキャッシュリード要求がオン（１）、キャッシュライト要求がオフ（０）、ｐｕｓｈ／ｐｏｐ修飾とＳＰ相対修飾のどちらもがオフ（０）であり、メモリに対するランダムリードの命令が実行された時の動作とその時の制御信号線の状態が示される。

【００８６】

図３（ｇ）はキャッシュリード要求がオン（１）、キャッシュライト要求がオフ（０）、ｐｕｓｈ／ｐｏｐ修飾がオフ（０）、ＳＰ相対修飾がオン（１）であり、メモリに対するＳＰ相対リード（ｌｏａｄ）命令が実行された時の動作とその時の制御信号線の状態が示される。

【００８７】

図３（ｈ）はキャッシュリード要求がオン（１）、キャッシュライト要求がオフ（０）、ｐｕｓｈ／ｐｏｐ修飾がオン（１）、ＳＰ相対修飾がオフ（０）であり、メモリに対するｐｏｐ命令が実行された時の動作とその時の制御信号線の状態が示される。

【００８８】

図３（ｉ）はキャッシュリード要求がオン（１）、キャッシュライト要求がオフ（０）、ｐｕｓｈ／ｐｏｐ修飾とＳＰ相対修飾のどちらもがオン（１）であり、通常このような状態は想定されておらず、このような状態は起こらない。

【００８９】

図３（ｊ）はキャッシュリード要求とキャッシュライト要求のどちらもがオン（１）であり、ｐｕｓｈ／ｐｏｐ修飾とＳＰ相対修飾の如何に関わらず、通常このような状態は想定されておらず、このような状態は起こらない。

【００９０】

図４は、本実施の形態のキャッシュユニット１１７のライン構成図であり、各ラインが有するデータ領域と制御に使われる制御ビット（フラグ）が示される。

【００９１】

図４に示すように本実施の形態のキャッシュユニット１１７の各ラインは、１６バイト（４ワード、１２８ビット）からなるデータ領域に加え、２０ビットからなるタグ、各１ビットのｖａｌｉｄフラグ、ｄｉｒｔｙフラグ、ｓｔａｃｋフラグを有している。

【００９２】

これら３つのフラグの取り得る組み合わせとその時の状態が示す意味を図５に示す。

【００９３】

以下にこれらの図を使い、本実施の形態がライトバックモード時に行うキャッシュ制御について説明する。

【００９４】

ＭＰＵ１１３がランダムリード命令を実行すると、図１に示すＭＰＵ１１３はキャッシュリード要求をオンにし、同時にリードしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力する。

【００９５】

キャッシュユニット１１７はこれらの信号を入力し、キャッシュビジー応答をオンにする。ＭＰＵ１１３はキャッシュビジー応答がオンになったことを確認してキャッシュリード要求をオフにする。

【００９６】

ＭＭＵ１１５は仮想アドレスバスの上位２０ビットを物理アドレスの上位２０ビットに変換し、実アドレスバスに出力する。仮想アドレスバスの下位１２ビットはそのまま実アドレスバスに出力される。

【００９７】

キャッシュユニット１１７はキャッシュリード要求信号を受け取るとまず、キャッシュビジー応答信号をオンにする。

【００９８】

次に実アドレスの中位８ビットをキーとして各ウエイ中の１つのセットを選択し、選択されたセットのタグの値を読み出す。本実施の形態のキャッシュユニットは２５６セット、４ウエイの構成を仮定しており、その構成を図１２に示す。

【００９９】

例えばアドレスバスの上位２０ビットが２進数表現で「０００１　１００１　１１００　００００　１１１１（１６進数表現で０ｘ１９Ｃ０Ｆ）」であり、例えばアドレスバスの中位８ビットが２進数表現で「００１０　１０００（１６進数表現で０ｘ２８、１０進数表現で『４０』）」であったとする。

【０１００】

この例の時には、『セット４０』に含まれる４つのラインのいずれかのラインがキャッシュに使用する可能性のあるラインとして選択される。

【０１０１】

しかしかしながら必ずしもこの構成に限らなければならないものではなく、その他のどのような構成でも本発明を有効に実施することができるが、これについては本発明の本質的な部分と直接関係しないので説明を省略する。

【０１０２】

次にキャッシュユニット１１７は実アドレスバスの上位２０ビットと読み出したタグの値（２０ビット）を比較し、一致したものがあればキャッシュはヒットしたと判断する。

【０１０３】

例えば上記の例では、アドレスバスの上位２０ビットが「０ｘ１９Ｃ０Ｆ」であるので、セット４０に含まれるラインに含まれるタグのいずれかの値がこれと同じ「０ｘ１９Ｃ０Ｆ」であるならばキャッシュがヒットしたと判断される。

【０１０４】

このキャッシュがヒットしたと判断された時の動作を図６（ｂ）～（ｇ）「動

作」の左欄「ランダムリード」に示す。

【０１０５】

キャッシュがヒットし、且つこのラインのｖａｌｉｄフラグが１の時の動作を図６（ｂ）～（ｅ）に示す。この時の状態は図５（ｅ）～（ｈ）に相当し、この時のラインの内容は主記憶装置１３１の内容と一致しているか主記憶装置１３１の内容よりも新しいので、このラインに含まれる全１６バイトから実アドレスバスの下位４ビットに該当するデータを読み出し、３２ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【０１０６】

またこの時、ｓｔａｃｋフラグが１ならば、図５（ｆ）、（ｈ）の状態に相当し、図６（ｃ）、（ｅ）に示す通り、ｓｔａｃｋフラグを０にする。

【０１０７】

ＭＰＵ１１３はキャッシュビジー応答信号がオフになったことを検出して、データバスの内容を読み込む。

【０１０８】

キャッシュはヒットしたがｖａｌｉｄフラグが０の時の動作を図６（ｆ）、（ｇ）に示す。この時の状態は図５（ａ）、（ｂ）に相当し、たまたまアドレスバスの上位２０ビットの値とタグの内容は一致したが、既にラインは開放されており、実際にはキャッシュされていなかったことを示している。

【０１０９】

即ちこの状態では、主記憶装置１３１の内容の方がラインの内容よりも新しいので、対応する主記憶装置１３１の内容をキャッシュメモリに読み込むリフィルを行うことが必要になる。

【０１１０】

リフィルを行った後、ｖａｌｉｄフラグを１にセットし、もしもｓｔａｃｋフラグが１になっていれば０にセットする。

【０１１１】

以上の動作によって、図６（ｂ）と同じ状態になるので、以後図６（ｂ）と同様に、このラインに含まれる全１６バイトから実アドレスバスの下位４ビットに

該当するデータを読み出し、３２ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【０１１２】

リフィルの詳細については次のキャッシュミスの所で説明する。

【０１１３】

一方、キャッシュユニット１１７が実アドレスバスの上位２０ビットと読み出したタグの値（２０ビット）を比較した結果、選択したセットに含まれるいずれのウエイにも一致したものがなければキャッシュはヒットしなかった（キャッシュミス）と判断する。

【０１１４】

この時の動作を図６（ａ）「動作」の左欄「ランダムリード」に示す。

【０１１５】

まず最初に、同一セット内にある４つのラインから１つを選択し、ラインの開放を行う。

【０１１６】

開放するラインの選択には例えば、ＬＲＵアルゴリズム（Ｌｅａｓｔ　Ｒｅｃｅｎｔｌｙ　Ｕｓｅｄ）」を使用する。

【０１１７】

これを実行するのが図１２の各セットに含まれる「ＬＲＵ式リプレーサ」である。

【０１１８】

このアルゴリズムは名前の通り、最近、最も使われていないもの（使われる頻度が少なかったもの）を開放する方法であるが、その内容は本発明の本質的部分と直接関係しないので説明を省略する。

【０１１９】

また必ずしもこのアルゴリズムに限るものではなく、他のアルゴリズムで開放するラインを決定しても勿論構わない。

【０１２０】

次にこのようにして選択したラインの開放を行う。ラインを開放する動作は各

ラインの状態、即ちその時そのラインの各フラグの値によって示される状態によって異なり、この内容は図７「動作」の右欄に示している。

【０１２１】

図７（ａ）の状態ではそのラインが主記憶装置１３１のキャッシュとして割り付けられていない、即ち開放されていることを示すので、改めで開放する必要はない。この状態は通常ｖａｌｉｄフラグが０であることで示される。

【０１２２】

図７（ｂ）、（ｃ）の状態では、図５（ｅ）、（ｆ）に示すように、ラインの内容と主記憶装置１３１の内容は一致しているので、ライトバックする必要はなく、只そのラインが開放されたことを示すためにｖａｌｉｄフラグを０にセットするだけで良い。また図７（ｃ）のようにｓｔａｃｋフラグが１になっており、ｓｔａｃｋに割り付けられていたことが示されているならば、そのラインの開放に際し、ｓｔａｃｋフラグも０にセットする。

【０１２３】

図７（ｄ）、（ｅ）の状態では、ｄｉｒｔｙフラグが１になっており、図５（ｇ）、（ｈ）に示すようにそのラインの内容は主記憶装置１３１の内容よりも新しく更新されているので、ラインの内容を主記憶装置１３１にライトバックすることが必要になる。

【０１２４】

このライトバックは図１に示す転送制御ユニット１１９を使い、転送データバスと外部データバスを介して高速且つ自動的に行われるが、この方法についても本発明の本質と直接関係しないので説明を省略する。

【０１２５】

ライトバックを行った後、一旦このラインが開放されたことを示すために、ｖａｌｉｄフラグとｄｉｒｔｙフラグを０にセットする。

【０１２６】

図７（ｅ）に示すｖａｌｉｄフラグが１、ｄｉｒｔｙフラグが１で、ｓｔａｃｋフラグも１の時は、図５（ｈ）の状態に相当し、このラインがスタックに割り付けられていたことが示されているので、この場合は前記と同様、ラインを開放

するに際しスタックへの割り付けを解除すると共にｓｔａｃｋフラグも０にセットする。

【０１２７】

図７（ｆ）、（ｇ）に示すｖａｌｉｄフラグが０の時は、図５（ａ）～（ｆ）の状態に相当し、たまたまタグの値は一致したが、既にこのラインはキャッシュとして使われていない（開放されている）のでライトバックを行う必要はない。もしもｓｔａｃｋフラグが１になっておれば、ラインの開放に際しｓｔａｃｋフラグを０にセットすることだけが行われる。

【０１２８】

以上で１つのラインの開放が終り、次に、この開放したラインをキャッシュとして利用するために、要求されたアドレスを含む１６バイトの主記憶装置１３１の内容がこのラインにリフィルされる。このリフィルも、図１に示す転送制御ユニット１１９を使い、転送データバスと外部データバスを介して高速且つ自動的に行われるが、この方法についても本発明の本質と直接関係しないので説明を省略する。

【０１２９】

そしてこのラインのタグに実アドレスの上位２０ビットを書き込み、ｖａｌｉｄフラグを１、ｄｉｒｔｙフラグを０、ｓｔａｃｋフラグを０、即ち図５（ｅ）の状態にセットする。

【０１３０】

ランダムリードの実行に際し、この状態は図６（ｂ）の状態と同じであるので、以後図６（ｂ）と同じ動作が行われる。

【０１３１】

即ち、このラインに含まれる全１６バイトから実アドレスバスの下位４ビットに該当するデータを読み出し、３２ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【０１３２】

ＭＰＵ１１３はキャッシュビジー応答信号がオフになったことを見て、データバスの内容を読み込む。

【０１３３】

ＭＰＵ１１３がランダムライト命令を実行すると、図１に示すＭＰＵ１１３はキャッシュライト要求をオンにし、同時にライトしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力し、更にライトしたいデータを３２ビット幅のデータバスに出力する。

【０１３４】

キャッシュユニット１１７はこれらの信号を入力し、キャッシュビジー応答をオンにする。ＭＰＵ１１３はキャッシュビジー応答がオンになったことを確認してキャッシュライト要求をオフにする。

【０１３５】

続けてランダムリードと同様の動作によって、セットとウエイの選択と、キャッシュヒットまたはキャッシュミスの判断が行われる。これについては既に説明したのでここでは説明を省略する。

【０１３６】

キャッシュがヒットしたと判断された時の動作を、図６（ｂ）～（ｇ）「動作」の右欄「ランダムライト」に示す。

【０１３７】

キャッシュがヒットし、且つこのラインのｖａｌｉｄフラグが１の時の動作を図６（ｂ）～（ｅ）に示す。この時の状態は図５（ｅ）～（ｈ）に相当し、この時のラインの内容は主記憶装置１３１の内容と一致しているか主記憶装置１３１の内容よりも新しいので、このラインの実アドレスバス下位４ビットに相当する領域にデータバスの内容を書き込み、まだｄｉｒｔｙフラグが１になっていなければ図６（ｂ）、（ｃ）に示すように、ｄｉｒｔｙフラグを１にセットし、キャッシュビジー応答信号をオフにする。

【０１３８】

またこの時、ｓｔａｃｋフラグが１ならば、図５（ｆ）、（ｈ）の状態に相当し、図６（ｃ）、（ｅ）に示す通り、ｓｔａｃｋフラグを０にする。

【０１３９】

ＭＰＵ１１３はキャッシュビジー応答信号がオフになったことを見て、ランダ

ムライトの終了したことを知る。

【０１４０】

キャッシュはヒットしたがｖａｌｉｄフラグが０の時の動作を図６（ｆ）、（ｇ）に示す。この時の状態は図５（ａ）、（ｂ）に相当し、たまたまアドレスバスの上位２０ビットの値とタグの内容は一致したが、既にラインは開放されており、実際にはキャッシュされていなかったことを示している。

【０１４１】

即ち、主記憶装置１３１の内容の方がラインの内容よりも新しいので、対応する主記憶装置１３１の内容をキャッシュメモリに読み込むリフィルを行うことが必要になる。リフィルを行った後、ｖａｌｉｄフラグを１にセットし、このラインの実アドレスバス下位４ビットに該当する領域にデータバスの内容を書き込み、ｄｉｒｔｙフラグを１にセットする。

【０１４２】

もしも図５（ｂ）の状態に相当し、ｓｔａｃｋフラグが１になっていれば、図６（ｇ）に示す通り、ラインのスタックへの割り付けを解除すると共にｓｔａｃｋフラグを０にセットする。

【０１４３】

一方、キャッシュミスと判断された時の動作を図６（ａ）「動作」の右欄「ランダムライト」に示す。

【０１４４】

新しくキャッシュに使用するラインを確保するためにアドレスバスの中位８ビットで選択したセットから、ＬＲＵ式リプレーサが開放するラインを決定して、決定したラインを開放し、開放したラインを新たにキャッシュとして使用するために、リフィルを行う。

【０１４５】

この動作は既に説明したランダムリードと同じであるからここでは説明を省略する。

【０１４６】

これらの動作によってラインの状態は主記憶装置１３１の内容と一致し、図６

（ｂ）の状態になるので、以後図６（ｂ）と同じ動作を行う。

【０１４７】

即ち、このラインの実アドレスバス下位４ビットに該当する領域にデータバスの内容を書き込み、キャッシュビジー応答信号をオフにする。

【０１４８】

ＭＰＵ１１３はキャッシュビジー応答信号がオフになったことを見て、ランダムライトの終了したことを知る。

【０１４９】

ＭＰＵ１１３がＳＰ相対リード命令を実行すると、ＭＰＵ１１３はキャッシュリード要求をオンにし、同時にリードしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力する。

【０１５０】

同時にＭＰＵ１１３はＳＰ相対修飾信号をオンにする。これによってキャッシュユニット１１７はＳＰ相対リード／ライト命令が実行されたことを知る。この点において、前記ランダムリード実行時と異なる。

【０１５１】

キャッシュユニット１１７はこれらの信号を入力し、キャッシュビジー応答をオンにする。ＭＰＵ１１３はキャッシュビジー応答がオンになったことを確認してキャッシュリード要求をオフにする。

【０１５２】

ＭＭＵ１１５は仮想アドレスバスの上位２０ビットを物理アドレスの上位２０ビットに変換し、実アドレスバスに出力する。仮想アドレスバスの下位１２ビットはそのまま実アドレスバスに出力される。

【０１５３】

キャッシュユニット１１７はキャッシュリード要求信号を受け取るとまず、キャッシュビジー応答信号をオンにする。

【０１５４】

次に実アドレスの中位８ビットをキーとして各ウエイ中の１つのセットを選択し、選択されたセットのタグの値を読み出して、キャッシュのヒット、ミスを判

断する。これらの動作はランダムリードと同様である。

【０１５５】

キャッシュがヒットし、且つこのラインのｖａｌｉｄフラグが１ならばこのラインから実アドレスバスの下位４ビットに該当するデータを読み出し、３２ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【０１５６】

この時の動作を図８（ｂ）～（ｅ）「動作」の左欄「ＳＰ相対リード」に示す。

【０１５７】

ＭＰＵ１１３はキャッシュビジー応答信号がオフになったことを見て、データバスの内容を読み込む。

【０１５８】

キャッシュはヒットしたがｖａｌｉｄフラグが０ならば、前記の通り、たまたまアドレスバスの上位２０ビットの値とあるラインのタグの内容が一致はしたが、既にそのラインは開放されており、キャッシュされていなかったことを示しているので、リフィルを行った後、ｖａｌｉｄフラグを１にセットし、その後図８（ｂ）と同じ動作が行われる。この時の動作を図８（ｆ）、（ｇ）の左欄「ＳＰ相対リード」に示す。

【０１５９】

一方、キャッシュユニット１１７が実アドレスバスの上位２０ビットと読み出したタグの値（２０ビット）を比較した結果、アドレスバスの中位８ビットの値によって選択したセット中のいずれのウエイにも一致したものがなければキャッシュはヒットしなかった（キャッシュミス）と判断し、同一セット内にある４つのラインから１つを選択してラインの開放を行う。これについても前記ランダムリードと同じであるから説明を省略する。

【０１６０】

その後、開放したラインに対して該当する主記憶装置１３１からその内容のリフィルを行い、リフィルを行った後ｖａｌｉｄフラグを１にセットし、その後図８（ｂ）と同じ動作が行われる。

【０１６１】

この時の動作を図８（ａ）の左欄「ＳＰ相対リード」に示す。

【０１６２】

図８（ａ）～（ｇ）に示すいずれの状態でもｓｔａｃｋフラグの設定は行わないので、ｓｔａｃｋフラグの状態は元の状態のまま維持される。これについてもランダムリードとはその動作を異にしている。

【０１６３】

ＭＰＵ１１３がＳＰ相対ライト命令を実行すると、ＭＰＵ１１３はキャッシュライト要求をオンにし、同時にリードしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力し、更にライトしたいデータを３２ビット幅のデータバスに出力する。

【０１６４】

同時にＭＰＵ１１３はＳＰ相対修飾信号をオンにする。これによってキャッシュユニット１１７はＳＰ相対リード／ライト命令が実行されたことを知る。この点において、前記ランダムリード実行時とは異なり、前記ＳＰ相対リード命令と同じである。

【０１６５】

キャッシュユニット１１７はこれらの信号を入力し、キャッシュビジー応答をオンにする。ＭＰＵ１１３はキャッシュビジー応答がオンになったことを確認してキャッシュリード要求をオフにする。

【０１６６】

続けてＳＰ相対リードと同様の動作によって、セットとウエイの選択と、キャッシュヒットまたはキャッシュミスの判断が行われる。これについては前記ランダムリードと同じである。

【０１６７】

キャッシュがヒットしたと判断された時の動作を、図８（ｂ）～（ｇ）「動作」の右欄「ＳＰ相対ライト」に示す。

【０１６８】

キャッシュがヒットし且つこのラインのｖａｌｉｄフラグが１ならば、このラ

インの実アドレスバス下位４ビットに相当する領域にデータバスの内容を書き込み、まだｄｉｒｔｙフラグが１になっていなければ図８（ｂ）、（ｃ）に示すように、ｄｉｒｔｙフラグを１にセットし、キャッシュビジー応答信号をオフにする。

【０１６９】

この時の動作を図８（ｂ）～（ｅ）「動作」の右欄「ＳＰ相対ライト」に示す。

【０１７０】

ＭＰＵ１１３はキャッシュビジー応答信号がオフになったことを見て、ＳＰ相対ライトの終了したことを知る。

【０１７１】

キャッシュはヒットしたがｖａｌｉｄフラグが０ならば、前記の通りたまたまアドレスバスの上位２０ビットの値とあるラインのタグの内容が一致はしたが、既にそのラインは開放されており、キャッシュされていなかったことを示しているので、リフィルを行った後、ｖａｌｉｄフラグを１にセットし、その後図８（ｂ）と同じ動作が行われる。この時の動作を図８（ｆ）、（ｇ）の右欄「ＳＰ相対ライト」に示す。

【０１７２】

一方、キャッシュユニット１１７が実アドレスバスの上位２０ビットと読み出したタグの値（２０ビット）を比較した結果、アドレスバスの中位８ビットの値によって選択したセット中のいずれのウエイにも一致したものがなければキャッシュはヒットしなかった（キャッシュミス）と判断し、同一セット内にある４つのラインから１つを選択してラインの開放を行う。

【０１７３】

その後、開放したラインに対して該当する主記憶装置１３１からその内容のリフィルを行い、リフィルを行った後、ｖａｌｉｄフラグを１にセットし、その後図８（ｂ）と同じ動作が行われる。この時の動作を図８（ａ）の右欄「ＳＰ相対ライト」に示す。

【０１７４】

図８（ａ）～（ｇ）に示すいずれの状態でもｓｔａｃｋフラグの設定は行わないので、ｓｔａｃｋフラグの状態は元の状態のまま維持される。この点についても前記ランダムライトとは異なり、前記ＳＰ相対リードと同じである。

【０１７５】

ＭＰＵ１１３がｐｏｐ命令を実行すると、ＭＰＵ１１３はキャッシュリード要求をオンにし、同時にｐｏｐしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力する。

【０１７６】

同時にＭＰＵ１１３はｐｕｓｈ／ｐｏｐ修飾信号をオンにする。この点において、前記ランダムリードやＳＰ相対リード実行時と異なる。これによってキャッシュユニット１１７はｐｕｓｈ／ｐｏｐ命令が実行されたことを知る。

【０１７７】

キャッシュユニット１１７はこれらの信号を入力し、キャッシュビジー応答をオンにする。ＭＰＵ１１３はキャッシュビジー応答がオンになったことを確認してキャッシュリード要求とｐｕｓｈ／ｐｏｐ修飾信号をオフにする。

【０１７８】

ＭＭＵ１１５は仮想アドレスバスの上位２０ビットを物理アドレスの上位２０ビットに変換し、実アドレスバスに出力する。仮想アドレスバスの下位１２ビットはそのまま実アドレスバスに出力される。

【０１７９】

キャッシュユニット１１７はキャッシュリード要求信号を受け取るとまず、キャッシュビジー応答信号をオンにする。

【０１８０】

次に実アドレスの中位８ビットをキーとして各ウエイ中の１つのセットを選択し、選択されたセットのタグの値を読み出して、キャッシュのヒット、ミスを判断する。これらの動作はランダムリードやＳＰ相対リードと同様であるから説明を省略する。

【０１８１】

実アドレスバスの上位２０ビットの値と選択したセット中の或るラインのタグ

の内容とが一致し、且つこのラインのｖａｌｉｄフラグが１ならばキャッシュはヒットしたと判断し、図１１（ｂ）～（ｅ）の動作を行う。

【０１８２】

実アドレスバスの第３ビット（下の桁から第４番目のビット、即ち２の３乗の桁を表すビット、以後同様）と第２ビット（下の桁から第３番目のビット、即ち２の２乗の桁を表すビット、以後同様）の両方が１ならば、図１１（ｂ）～（ｅ）の動作左欄「最上位アドレスからｐｏｐ」の動作を行う。

【０１８３】

即ち、このラインから実アドレスバスの下位４ビットに該当するデータを読み出し、３２ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【０１８４】

次に、このラインのｖａｌｉｄフラグを０にし、このラインを開放する。最上位アドレスからデータがｐｏｐされたことによってこのスタックが空になったためである。

【０１８５】

この時、ｖａｌｉｄフラグを０にセットして、ラインを開放するが、ライトバックは行わない。このラインの内容は再度参照されることはなく、ライトバックして主記憶に保存する必要がないからである。

【０１８６】

また図１１（ｂ）、（ｄ）の動作左欄「最上位アドレスからｐｏｐ」に示す通り、このラインのｓｔａｃｋフラグがオフになっていればオンにする。このラインがスタックとして使用されていたことを記憶しておくためである。

【０１８７】

また図１１（ｄ）、（ｅ）の動作左欄「最上位アドレスからｐｏｐ」に示す通り、このラインのｄｉｒｔｙフラグがオンになっていればオフにする。このラインは開放されるので、ｄｉｒｔｙフラグは意味を持たないからである。

【０１８８】

実アドレスバスの第３ビットと第２ビットのいずれか少なくとも一方が０なら

ば、図１１（ｂ）～（ｅ）の動作右欄「最上位アドレス以外からｐｏｐ」の動作を行う。

【０１８９】

即ち、このラインから実アドレスバスの下位４ビットに該当するデータを読み出し、３２ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【０１９０】

ＣＰＵはキャッシュビジー応答信号がオフになったことを見て、３２ビット幅データバスの情報を読み込む。フラグの操作等は行わない。

【０１９１】

実アドレスバスの上位２０ビットの値と選択したセット中の或るラインのタグの内容とが一致したが、このラインのｖａｌｉｄフラグが０ならば、図１１（ｆ）（ｇ）の動作を行う。

【０１９２】

この時は、たまたま選択したセット中の或るラインのタグの内容と実アドレスバスの上位２０ビットの値が一致したが、既にこのラインは開放されていたので、リフィルを行う。

【０１９３】

実アドレスバスの第３ビットと第２ビットの両方が１ならば、図１１（ｆ）（ｇ）の動作左欄「最上位アドレスからｐｏｐ」の動作を行う。

【０１９４】

即ち、このラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを行い、ｖａｌｉｄフラグを１にする。そしてこのラインから実アドレスバスの下位４ビットに該当するデータを読み出し、３２ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【０１９５】

もしもこの時ｓｔａｃｋフラグが０ならば１にする（図１１（ｆ）動作左欄「最上位アドレスからｐｏｐ」）。

【０１９６】

更に、ｖａｌｉｄフラグを０に戻す（図１１（ｆ）（ｇ）動作左欄「最上位アドレスからｐｏｐ」）。最上位アドレスからのｐｏｐによって、このラインを開放するためである。

【０１９７】

実アドレスバスの第３ビットと第２ビットの何れか少なくとも一方が０ならば、図１１（ｆ）（ｇ）の動作右欄「最上位アドレス以外からｐｏｐ」の動作を行う。

【０１９８】

即ち、このラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを実行し、ｖａｌｉｄフラグを１にする。そしてこのラインから実アドレスバスの下位４ビットに該当するデータを読み出し、３２ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。フラグの操作等は行わない。

【０１９９】

実アドレスバスの上位２０ビットの値と選択したセット中のラインのタグの内容とが一致するものがなければ、キャッシュはミスしたと判断し、図１１（ａ）の動作を行う。

【０２００】

即ち、このセットからＬＲＵアルゴリズムによって１つのラインを開放し、そのラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを実行する。そしてｖａｌｉｄフラグを１に、ｄｉｒｔｙフラグを０にセットし、このラインから実アドレスバスの下位４ビットに該当するデータを読み出し、３２ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする（図１１（ａ））。

【０２０１】

更に、実アドレスバスの第３ビットと第２ビットの両方が１ならば、ｓｔａｃｋフラグを１にする（図１１（ａ）動作左欄「最上位アドレスからｐｏｐ」）。

【０２０２】

更にまたこの時は、ｖａｌｉｄフラグを０に戻す（図１１（ａ）動作左欄「最

上位アドレスからｐｏｐ」）。最上位アドレスからのｐｏｐによって、このラインを開放するためである。

【０２０３】

以上の動作の後或いは途中、もしも転送ビジー応答信号がオフならば、転送データバスが空いているので、もうすぐ必要となる可能性のあるメモリについて予めリフィルを実行する。

【０２０４】

具体的には、ラインに含まれるタグの値とそのラインを含むセットによって示される実アドレスの上位２８ビットが、現在のＳＰの実アドレスの上位２８ビットよりも１または２大きなラインを１つ見付け、図１０の動作右欄「ＳＰが下方から近付いた」に示す動作を行う。

【０２０５】

このようなラインを見付けることができ、且つそのラインのｖａｌｉｄフラグが１ならば何も行わない（図１０動作右欄（ｂ）～（ｅ））。

【０２０６】

このようなラインを見付けることはできたが、そのラインのｖａｌｉｄフラグが０であり、且つそのラインがスタックに割り当てられていれば、そのラインのリフィルを行い、ｖａｌｉｄフラグを１にする（図１０動作右欄（ｇ））。

【０２０７】

このようなラインを見付けることはできたが、そのラインのｖａｌｉｄフラグが０であり、且つそのラインがスタックに割り当てられていなければ、何も行わない（図１０動作右欄（ｆ））。

【０２０８】

またこのようなラインを見付けることができなければ、現在のＳＰのすぐ上にあり、すぐにｐｏｐによって読み出される可能性の高い主記憶装置の領域がキャッシュされていないことになるので、実アドレスの上位２８ビットが現在のＳＰの実アドレスの上位２８ビットよりも１大きく、下位４ビットは０である主記憶装置の領域についてキャッシュ上にリフィルを実行する（図１０動作右欄（ａ））。

【０２０９】

具体的には、現在のＳＰの実アドレスの上位２８ビットよりも１大きくなアドレスに対応するセットで、ＬＲＵによって開放するラインを決定し、決定したラインの開放を行い、そのラインに実アドレスの上位２８ビットが現在のＳＰの実アドレスの上位２８ビットよりも１大きく、下位４ビットは０である主記憶装置の領域のリフィルを実行する。そしてそのラインのｖａｌｉｄフラグを１に設定する。

【０２１０】

また同時に、ラインに含まれるタグの値とそのラインを含むセットによって示される実アドレスの上位２８ビットが、現在のＳＰの実アドレスの上位２８ビットよりも小さなラインを捜し、もしもそのようなラインを見付けることができたら、図７の動作左欄「ＳＰが上方に移動」に示す動作を行う。

【０２１１】

即ち、このようなラインを見付けることができ、更にそのラインのｖａｌｉｄフラグ、ｄｉｒｔｙフラグ、ｓｔａｃｋフラグがいずれも１であったならば、そのラインのライトバックを行い、ｄｉｒｔｙフラグを０にする（図７動作左欄（ｅ））。

【０２１２】

このようなラインを見付けることができなかったか、或いは見付けることができてもｖａｌｉｄフラグ、ｄｉｒｔｙフラグ、ｓｔａｃｋフラグがいずれかが０であったならば何も実行しない（図７動作左欄（ａ）～（ｄ）（ｆ）（ｇ））。

【０２１３】

ＭＰＵ１１３がｐｕｓｈ命令を実行すると、ＭＰＵ１１３はキャッシュライト要求信号をオンにし、同時にリードしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力し、更にライトしたいワードデータを３２ビット幅のデータバスに出力する。

【０２１４】

同時にＭＰＵ１１３はｐｕｓｈ／ｐｏｐ修飾信号をオンにする。この点において、前記ランダムリードやＳＰ相対リード実行時とは異なり、前記ｐｏｐ命令時

と同じである。これによってキャッシュユニット１１７はｐｕｓｈ／ｐｏｐ命令が実行されたことを知る。

【０２１５】

キャッシュユニット１１７はこれらの信号を入力し、キャッシュビジー応答信号をオンにする。ＭＰＵ１１３はキャッシュビジー応答信号がオンになったことを確認してキャッシュライト要求信号とｐｕｓｈ／ｐｏｐ修飾信号をオフにする。

【０２１６】

続けてｐｏｐと同様の動作によって、セットとウエイの選択と、キャッシュヒットまたはキャッシュミスの判断が行われる。これらの動作はランダムライトやＳＰ相対ライトと同様であるから説明を省略する。

【０２１７】

実アドレスバスの上位２０ビットの値と選択したセット中の或るラインのタグの内容とが一致し、且つこのラインのｖａｌｉｄフラグが１ならばキャッシュはヒットしたと判断し、該当するラインにおいて、図９（ｂ）～（ｅ）の動作左欄「最上位アドレスにｐｕｓｈ」または動作右欄「最上位アドレス以外にｐｕｓｈ」の動作を行う。

【０２１８】

実アドレスバスの第３ビットと第２ビットの両方が１ならば、図９（ｂ）～（ｅ）の動作左欄「最上位アドレスにｐｕｓｈ」の動作を行う。

【０２１９】

即ち、３２ビット幅のデータバスからライトしたいワードデータを読み込み、このラインの実アドレスバスの下位４ビットに該当する領域にライトし、キャッシュビジー応答信号をオフにする（図９（ｂ）～（ｅ）動作左欄「最上位アドレスにｐｕｓｈ」）。

【０２２０】

次に、もしもこのラインのｄｉｒｔｙフラグが０ならば１にセットする（図９（ｂ）（ｃ）動作左欄「最上位アドレスにｐｕｓｈ」）。

【０２２１】

また、もしもこのラインのｓｔａｃｋフラグが０ならば１にセットする（図９（ｂ）（ｄ）動作左欄「最上位アドレスにｐｕｓｈ」）。

【０２２２】

ＣＰＵはキャッシュヒジー応答信号がオフになったことを見て、ｐｕｓｈ命令が完了したことを知る。

【０２２３】

実アドレスバスの第３ビットと第２ビットの何れか少なくとも一方が０ならば、図９（ｂ）～（ｅ）の動作右欄「最上位アドレス以外にｐｕｓｈ」の動作を行う。

【０２２４】

即ち、３２ビット幅のデータバスからライトしたいワードデータを読み込み、このラインの実アドレスバスの下位４ビットに該当する領域にライトし、キャッシュヒジー応答信号をオフにする（図９（ｂ）～（ｅ）動作右欄「最上位アドレス以外にｐｕｓｈ」）。

【０２２５】

次に、もしもこのラインのｄｉｒｔｙフラグが０ならば１にセットする（図９（ｂ）（ｃ）動作右欄「最上位アドレス以外にｐｕｓｈ」）。

【０２２６】

その他のフラグ操作は行わない。

【０２２７】

ＣＰＵはキャッシュヒジー応答信号がオフになったことを見て、ｐｕｓｈ命令が完了したことを知る。

【０２２８】

実アドレスバスの上位２０ビットの値と選択したセット中の或るラインのタグの内容とが一致したが、このラインのｖａｌｉｄフラグが０ならば、図９（ｆ）（ｇ）の動作を行う。

【０２２９】

この時は、たまたま選択したセット中の或るラインのタグの内容と実アドレスバスの上位２０ビットの値が一致したが、既にこのラインは開放されていたので

、リフィルを行う。

【０２３０】

実アドレスバスの第３ビットと第２ビットの両方が１ならば、図９（ｆ）（ｇ）の動作左欄「最上位アドレスにｐｕｓｈ」の動作を行う。

【０２３１】

即ち、このラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを行い、ｖａｌｉｄフラグを１にする。そして３２ビット幅のデータバスからライトしたいワードデータを読み込み、このラインの実アドレスバスの下位４ビットに該当する領域にライトし、キャッシュビジー応答信号をオフにする（図９（ｆ）（ｇ）の動作左欄「最上位アドレスにｐｕｓｈ」）。ＣＰＵはキャッシュビジー応答信号がオフになったことを見て、ｐｕｓｈ命令が完了したことを知る。

【０２３２】

ｄｉｒｔｙフラグは１にセットする（図９（ｆ）（ｇ）の動作左欄「最上位アドレスにｐｕｓｈ」）。

【０２３３】

もしもこの時ｓｔａｃｋフラグが０ならば１にする（図９（ｆ）動作左欄「最上位アドレスへｐｕｓｈ」）。

【０２３４】

実アドレスバスの第３ビットと第２ビットの何れか少なくとも一方が０ならば、図９（ｆ）（ｇ）の動作右欄「最上位アドレス以外へｐｕｓｈ」の動作を行う。

【０２３５】

即ち、このラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを実行し、ｖａｌｉｄフラグを１にする。そして、３２ビット幅のデータバスからライトしたいワードデータを読み込み、このラインの実アドレスバスの下位４ビットに該当する領域にライトし、キャッシュビジー応答信号をオフにする。ｄｉｒｔｙフラグは１にセットする。

【０２３６】

実アドレスバスの上位２０ビットの値と選択したセット中のラインのタグの内容とが一致するものがなければ、キャッシュはミスしたと判断し、図９（ａ）の動作を行う。

【０２３７】

即ち、このセットからＬＲＵアルゴリズムによって１つのラインを開放し、そのラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを実行する。そしてｖａｌｉｄフラグを１に、ｄｉｒｔｙフラグを０に、ｓｔａｃｋフラグを０にセットする。

【０２３８】

これによって図９（ｂ）と同じ状態になるので、以後図９（ｂ）の動作を実行する。

【０２３９】

以上の動作の後或いは途中、もしも転送ビジー応答信号がオフならば、転送データバスが空いているので、当分の間必要となる可能性の低いメモリについて予めライトバックを実行する。

【０２４０】

具体的には、ＭＭＵのアドレス変換の単位である、ラインに含まれるタグの値の方が現在のＳＰの実アドレスの上位２０ビットよりも大きいラインか、またはラインに含まれるタグの値とそのラインを含むセットによって示される実アドレスの上位２８ビットが、現在のＳＰの実アドレスの上位２８ビットよりも３以上大きなラインを、１つ見付け図１０の動作左欄「ＳＰが下方に遠ざかった」に示す動作を行う。

【０２４１】

このようなラインを見付けることはできたが、そのラインのｖａｌｉｄフラグとｄｉｒｔｙフラグとｓｔａｃｋフラグの何れか少なくとも１つが０ならば何も行わない（図１０動作左欄（ｂ）～（ｄ）（ｆ）（ｇ））。

【０２４２】

このようなラインを見付けることができ、且つそのラインのｖａｌｉｄフラグとｄｉｒｔｙフラグとｓｔａｃｋフラグの全てが１ならば、そのラインのライト

バックを行い、ｄｉｒｔｙフラグを０にする（図１０動作左欄（ｅ））。これによってもしもこのラインがＬＲＵアルゴリズムによって開放されることになった時、改めてライトバックを行う必要がなく、処理を高速化することができる。

【０２４３】

このようなラインを見付けることができなければ、何も行わない（図１０動作左欄（ａ））。

【０２４４】

（実施の形態２）

本発明の第２の実施の形態であるデータメモリキャッシュ制御装置の作用を、Ｃ言語で記述されたプログラムを実行する汎用ＣＰＵを例に、図１４～図１９を使って詳細に説明する。

【０２４５】

Ｃ言語で記述されたプログラムは、例えば上位アドレスから下位アドレス方向に向けて成長するスタック構造を使用する。

【０２４６】

汎用ＣＰＵ内に存在する専用スタックポインタ（ＳＰ）を使用し、同汎用ＣＰＵ内に存在する汎用レジスタをフレームポインタ（ＦＰ）として使用し、このスタック構造にアクセスを実行する。

【０２４７】

この時のスタック構造の一例を図１３に示す。この例には１つの関数の実行に関わるスタック構造だけを示している。

【０２４８】

実際には、既に実行された関数コール（呼び出し）の数に対応するスタック構造が、下方に積み重ねられた構造をしている。

【０２４９】

次に、この構造がＣ言語による新しい関数呼び出しでどのように生成（変化）されるかを説明する。

【０２５０】

Ｃ言語の関数が呼び出される時にはまず、関数を呼び出す側がスタックトップ

にｐｕｓｈ命令を使い、関数側での処理に必要な引き数を生成する。この引き数を関数に引き渡すためである。

【０２５１】

この引き数を生成した直後のスタック構造を図１４に示す。

【０２５２】

次に、関数呼び出しを実行する。この時、ｐｕｓｈ命令を実行して関数からの戻りアドレスをスタックに退避する。この関数呼び出し実行直後のスタック構造を図１５に示す。

【０２５３】

次にプログラムの実行は、呼び出された関数側に移行する。

【０２５４】

呼び出された関数側では最初に、ｐｕｓｈ命令を実行し、フレームポインタをも含めた関数内部で使用するレジスタをスタックに退避する。このレジスタ退避直後のスタック構造を図１６に示す。

【０２５５】

次に呼び出された関数では、スタックポインタの値をフレームポインタに設定し、スタックフレームを生成するために、スタックポインタからスタックフレームのサイズを減算する。これによってスタックポインタはより下位アドレス側に移動し、このスタックポインタとフレームポインタの間にスタックフレームが生成される。

【０２５６】

スタックフレームが生成された直後のスタック構造を図１７に示す。

【０２５７】

呼び出された関数側では、次に、フレームポインタ相対リード・ライト命令、スタックポインタ相対リード・ライト命令、ランダムリード・ライト命令等を実行して、引き数にアクセスしたり、スタックフレームにアクセスし、必要な情報にアクセスして関数の処理を実行する。

【０２５８】

また必要なら、他の関数呼び出しを実行することもある。

【０２５９】

呼び出された関数側では、関数の処理を終了すると、使用していたスタックフレームを破棄するため、スタックポインタにスタックフレームのサイズを加算する。

【０２６０】

そしてｐｏｐ命令を実行し、退避していたレジスタを復元する。

【０２６１】

この時のスタック構造は、関数呼び出し直後のスタック構造と完全に同じ状態に復元され、図１５に示される。

【０２６２】

次にｐｏｐ命令を実行して、呼び出し側の関数へ戻るためのアドレスを取り出し、リタン命令によって、呼び出された関数から呼び出した関数にプログラム実行を復帰させる。

【０２６３】

この時のスタック構造は、関数に引き渡すための引き数を生成した直後のスタック構造と完全に同一となり、図１４に示す。

【０２６４】

復帰した呼び出し関数側では、引き数列を廃棄するため、スタックポインタに引き数列の大きさを加算する。例えば図１４の例では２掛けるワードサイズに相当する値である。

【０２６５】

以上の処理によって、関数呼び出し処理を開始する前のスタック構造に復帰する。この時のスタック構造は図１３に示す通りである。

【０２６６】

以上の関数呼び出しに関わる一連の処理における、スタックフレーム生成直後のキャッシュラインの状態を図１８に示す。

【０２６７】

図１８に示す通り、スタックフレーム以外の部分はスタックとして使用され、そのためにラインのｓｔａｃｋフラグは１が設定される。

【０２６８】

キャッシュラインの単位は１６バイトであり、スタック構造の単位は４バイトであるので、スタックフレームとレジスタ退避スタックとの境界がキャッシュラインの境界と一致するとは限らない。

【０２６９】

本発明によると、レジスタ退避スタックとスタックフレームとが同居しているラインでは、ｓｔａｃｋフラグが１に設定される。

【０２７０】

またスタックフレームと引き数列が同居しているラインではｓｔａｃｋフラグが０に設定される。

【０２７１】

関数処理実行中に、引き数列やスタックフレームに対するアクセスを実行すると、通常これらのアクセスはランダムリード・ライト命令によって行われるため、１に設定されていたｓｔａｃｋフラグも０に変化することが起ってくる。

【０２７２】

従って、レジスタ退避スタックとスタックフレームとが同居したラインや、引き数列と戻りアドレスが同居しているラインも、ｓｔａｃｋフラグは０になってくる。

【０２７３】

この時の各ラインのフラグの状態を図１９に示す。

【０２７４】

（実施の形態３）

本発明の第３の実施の形態であるデータメモリキャッシュ制御装置の作用を、Ｊａｖａ（Ｒ）言語で記述されたプログラムを実行する汎用ＣＰＵを例に、図２０～図２４を使って詳細に説明する。

【０２７５】

Ｃ言語で記述されたプログラムは、例えばＣ言語で記述されたインタープリタによって実行される。

【０２７６】

このインタープリタは、Ｃ言語で記述されたプログラムを実行するためのプログラムカウンタの他に、Ｊａｖａ（Ｒ）仮想マシンを実装するために必要なＪａｖａ（Ｒ）プログラムカウンタを有している。

【０２７７】

またスタック構造においても、Ｃ言語で記述されたプログラムを実行するためのスタック構造以外に、Ｊａｖａ（Ｒ）仮想マシンを実装するために必要なＪａｖａ（Ｒ）スタック構造を有している。

【０２７８】

Ｊａｖａ（Ｒ）スタック構造も例えば、上位アドレスから下位アドレス方向に向けて成長するスタック構造を使用することができる。

【０２７９】

このＪａｖａ（Ｒ）スタック構造では例えば、汎用ＣＰＵ内に存在する汎用レジスタをスタックポインタ（ＳＰ）として使用し、このＪａｖａ（Ｒ）スタック構造にアクセスを実行する。

【０２８０】

Ｊａｖａ（Ｒ）スタック構造の一例を図２０に示す。

【０２８１】

図２０には、１つのメソッド（Ｃ言語における関数に相当する）の処理に対応するＪａｖａ（Ｒ）スタック構造が示されている。

【０２８２】

実際には、これまでに発生したメソッド呼び出し（コール：ｃａｌｌ）に相当する数のスタック構造が、下方に積み重ねられた構造をしている。

【０２８３】

次に、このスタック構造が、Ｊａｖａ（Ｒ）言語による新しいメソッド呼び出しによってどのように生成（変化）されるかを説明する。

【０２８４】

最初に、新しいメソッドを呼び出すメソッド呼び出し側が、スタックトップにｐｕｓｈ命令を実行することによってメソッド内での処理に使用するための引き数を設定する。

【０２８５】

この引き数列生成直後のＪａｖａ（Ｒ）スタック構造を図２１に示す。

【０２８６】

この引き数列は、呼び出されるメソッド側での処理で必要となる値を、メソッド呼び出し側から呼び出されるメソッド側に引き渡すために使用される。

【０２８７】

次にメソッド呼び出しが実行される。

【０２８８】

メソッドからの戻りＪａｖａ（Ｒ）アドレスは、Ｃ言語スタックに退避される。このアドレスの退避はＪａｖａ（Ｒ）スタック構造とは直接関係しないので説明を省略する。

【０２８９】

このメソッド呼び出しによって、Ｊａｖａ（Ｒ）プログラムの制御は、呼び出されたメソッド側に移行する。

【０２９０】

呼び出されたＪａｖａ（Ｒ）メソッド側では、使用するレジスタの退避が必要であったならばこれらのレジスタの値をＣ言語スタックに退避する。このレジスタの退避もＪａｖａ（Ｒ）スタック構造とは直接関係しないので説明を省略する。

【０２９１】

次に、呼び出されたメソッドでは、自分自身が実行する処理で必要となるローカル変数のための領域を生成するために、Ｊａｖａ（Ｒ）スタックポインタの値からローカル変数の領域のサイズに相当する値を減算する。

【０２９２】

このローカル変数領域生成直後のＪａｖａ（Ｒ）スタック構造を図２２に示す。

【０２９３】

呼び出されたメソッドでは、その後、Ｊａｖａ（Ｒ）スタックポインタ相対アドレス指定である、ＳＰ相対リード・ライト命令によって引き数やローカル変数

にアクセスし、メソッドの処理を実行する。

【０２９４】

Ｊａｖａ（Ｒ）言語はスタック指向言語であるためＪａｖａ（Ｒ）命令でのオペランドは、必ずＪａｖａ（Ｒ）スタックポインタ相対として指定される。

【０２９５】

例えば加算（ａｄｄ）命令ではオペランドが明示的に指定されることはなく、Ｊａｖａ（Ｒ）スタックのトップとセカンドに自動的に設定される。

【０２９６】

従って、Ｊａｖａ（Ｒ）インタープリタはａｄｄ命令を実行するには、ｐｏｐ命令を２回実行することによってＪａｖａ（Ｒ）スタック中のオペランドスタックから２つの値を読み出し、加算を実行し、その結果をｐｕｓｈ命令を実行することによってＪａｖａ（Ｒ）スタック中のオペランドスタックのトップに格納する。

【０２９７】

この処理を実行中のＪａｖａ（Ｒ）スタック構造を図２３に示す。

【０２９８】

呼び出されたメソッドでは必要に応じて、更に他のメソッドを呼び出すこともある。

【０２９９】

呼び出されたメソッドでの処理が終了した時点で、Ｊａｖａ（Ｒ）スタック中のオペランドスタックは空になる。

【０３００】

そしてローカル変数領域と引き数列を破棄するため、Ｊａｖａ（Ｒ）スタックポインタにこれの合計サイズに相当する値を加算する。

【０３０１】

この時のＪａｖａ（Ｒ）スタック構造は、Ｊａｖａ（Ｒ）メソッド呼び出し処理を開始する前のＪａｖａ（Ｒ）スタック構造に復帰する。この時のＪａｖａ（Ｒ）スタック構造は図２０に示される。

【０３０２】

以上のＪａｖａ（Ｒ）メソッド呼び出し処理の実行における、処理実行中のキャッシュラインの状態を、図２４に示す。

【０３０３】

図２４に示す通り、このメソッド呼び出し処理のために生成したラインのｓｔａｃｋフラグは１に設定される。

【０３０４】

また、Ｊａｖａ（Ｒ）スタックポインタがスタックボトムから遠ざかることによって、スタックポインタから離れたラインで転送ビジー応答信号がオフの時間中にライトバックが実行され、そのラインのｄｉｒｔｙフラグが０に設定された状態をも図２４に示す。

【０３０５】

【発明の効果】

本特許出願に係る請求項２に記載の発明は、ＣＰＵからの連続したアドレスのメモリ領域に対する連続書き込み時にキャッシュミスが発生した時、次に書き込みが行われるアドレス方向のメモリ領域に対しては主記憶装置からキャッシュメモリへの読み込みを行わないので、高速化の効率を大きく改善することができる。

【０３０６】

本特許出願に係る請求項３に記載の発明は、ＣＰＵによる連続したアドレスのメモリ領域に対する連続読み込み時にキャッシュミスが発生した時、既に読み込みが行われたアドレス方向のメモリ領域に対してはキャッシュメモリから主記憶装置への書き込みを行わないので、高速化の効率を大きく改善することができる。

【０３０７】

本特許出願に係る請求項４に記載の発明は、ＣＰＵからの連続したアドレスのメモリ領域に対する連続書き込み時に、既に書き込みが行われたアドレス方向のメモリ領域で、現在書き込みが行われているメモリ領域とのアドレス距離が一定以上離れたメモリ領域に対して、バスの空き時間を使って、予めキャッシュメモリから主記憶装置への書き込みを行うので、高速化の効率を大きく改善すること

ができる。

【０３０８】

本特許出願に係る請求項５に記載の発明は、ＣＰＵによる連続したアドレスのメモリ領域に対する連続読み込み時に、今後読み込みが行われるアドレス方向のメモリ領域で、現在読み込みが行われているメモリ領域とのアドレス距離が一定以上近いメモリ領域に対して、バスの空き時間を使って、予め主記憶装置からキャッシュメモリへの読み込みを行うので、高速化の効率を大きく改善することができる。

【図面の簡単な説明】

【図１】

本発明の第１の実施の形態であるデータメモリキャッシュ制御装置のブロック構成を示す図

【図２】

ｐｕｓｈ、ｐｏｐ、ｌｏａｄ、ｓｔｏｒｅ命令の各命令形式を示す図

【図３】

ＭＰＵによって実行される命令とその動作と各制御線の状態との取り得る組み合わせを示す図

【図４】

本発明の第１の実施の形態のキャッシュユニット１１７のライン構成図

【図５】

ｖａｌｉｄフラグ、ｄｉｒｔｙフラグ、ｓｔａｃｋフラグの取り得る組み合わせとその時の状態が示す意味を示す図

【図６】

ランダムリード・ランダムライト命令実行時の各ラインの状態に対応した動作の内容を示す図

【図７】

ラインの開放・ＳＰが上方に移動を実行時の各ラインの状態に対応した動作の内容を示す図

【図８】

ＳＰ相対リード・ＳＰ相対ライト命令実行時の各ラインの状態に対応した動作の内容を示す図

【図９】

ｐｕｓｈ命令実行時の各ラインの状態に対応した動作の内容を示す図

【図１０】

ＳＰが下方に遠ざかった・ＳＰが下方から近付いた実行時の各ラインの状態に対応した動作の内容を示す図

【図１１】

ｐｏｐ命令実行時の各ラインの状態に対応した動作の内容を示す図

【図１２】

本発明の第１の実施の形態のキャッシュユニットの構成を示す図

【図１３】

Ｃ言語のスタック構造を示す図

【図１４】

引き数生成直後のスタック構造を示す図

【図１５】

関数呼び出し直後のスタック構造を示す図

【図１６】

レジスタ退避直後のスタック構造を示す図

【図１７】

スタックフレーム生成直後のスタック構造を示す図

【図１８】

スタックフレーム生成直後のキャッシュラインの状態を示す図

【図１９】

ランダムアクセス後のキャッシュラインの状態を示す図

【図２０】

Ｊａｖａ（Ｒ）スタック構造を示す図

【図２１】

引き数生成直後のＪａｖａ（Ｒ）スタック構造を示す図

【図２２】

ローカル変数領域生成直後のＪａｖａ（Ｒ）スタック構造を示す図

【図２３】

関数処理中のＪａｖａ（Ｒ）スタック構造を示す図

【図２４】

関数処理中のキャッシュラインの状態を示す図

【図２５】

従来のスタックキャッシュ制御装置においてｐｕｓｈでキャッシュミスが発生した時に１単位の内容が主記憶装置からキャッシュメモリに読み込まれる状態を示す図

【図２６】

従来のスタックキャッシュ制御装置においてｐｏｐでキャッシュミスが発生した時に１単位の内容がキャッシュメモリから主記憶装置にライトバックされる状態を示す図

【図２７】

低いアドレスのメモリ領域に記憶されたデータが当分の間はポップによって読み出される可能性が非常に低い状態を示す図

【図２８】

スタックポインタの示すアドレスよりも低い値のアドレス領域はすぐに続けてポップによって読み出される可能性が非常に高い状態を示す図

【符号の説明】

１１３　演算ユニット（ＭＰＵ）

１１５　メモリマネジメントユニット（ＭＭＵ）

１１７　スタック対応データキャッシュユニット

１１９　転送制御ユニット

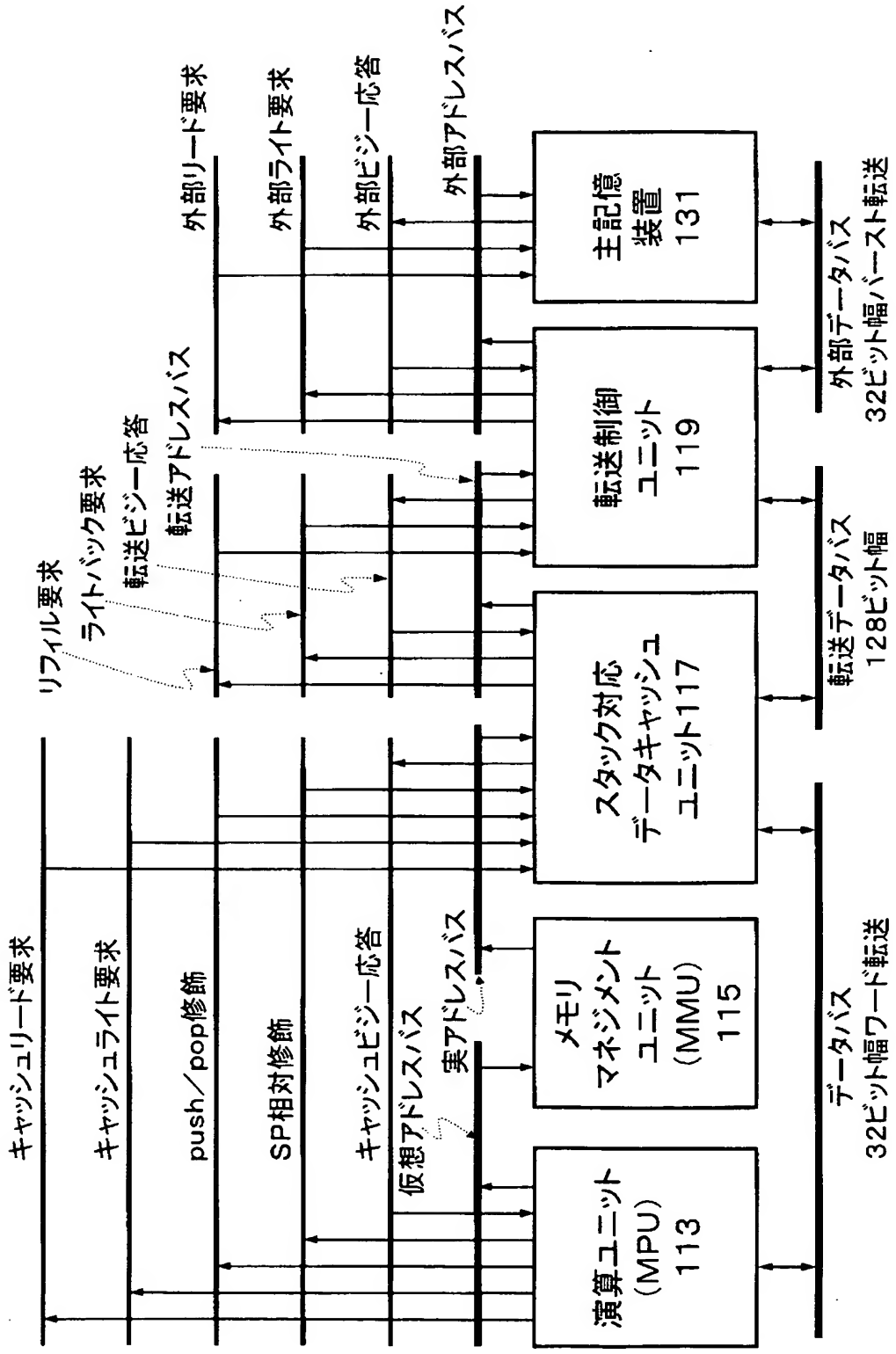
１３１　主記憶装置

９１０，９２０，９３０，９４０　スタックメモリ

９１３，９２３，９３３，９４３　スタックポインタ

【書類名】　図面

【図１】


キャッシュリード要求
キャッシュライト要求
push／pop修飾
SP相対修飾
キャッシュヒット応答
仮想アドレスバス
実アドレスバス
リフィル要求
ライトバック要求
転送ビジー応答
転送アドレスバス
外部リード要求
外部ライト要求
外部ビジー応答
外部アドレスバス
演算ユニット
(MPU)
113
メモリ
マネジメント
ユニット
(MMU)
115
スタック対応
データキャッシュ
ユニット117
転送制御
ユニット
119
主記憶
装置
131
データバス
32ビット幅ワード転送
転送データバス
128ビット幅
外部データバス
32ビット幅バースト転送

【図２】

	命令	動作
(a)	push　Rn, [−Rm]	Rm=Rm−4 [Rm]=Rn push／pop修飾=1
(b)	pop　[Rn+], Rm	push／pop修飾=1 Rm=[Rn] Rn=Rn+4
(c)	load　[Rn+オフセット], Rm	SP相対修飾=1 Rm=[Rn+オフセット]
(d)	store　Rn, [Rm+オフセット]	[Rm+オフセット]=Rn SP相対修飾=1

【図３】

	キャッシュ リード要求	キャッシュ ライト要求	push／ pop修飾	SP相対 修飾	命令／動作
(a)	0	0	X	X	—
(b)	0	1	0	0	ランダムライト
(c)	0	1	0	1	SP相対ライト
(d)	0	1	1	0	push
(e)	0	1	1	1	—
(f)	1	0	0	0	ランダムリード
(g)	1	0	0	1	SP相対リード
(h)	1	0	1	0	pop
(i)	1	0	1	1	—
(j)	1	1	X	X	—

【図４】

本発明の一実施の形態のキャッシュユニット117のライン構成図

	タグ	validフラグ	dirtyフラグ	stackフラグ	データ領域
ライン1	20ビット	1ビット	1ビット	1ビット	16バイト(4ワード)
ライン2	20ビット	1ビット	1ビット	1ビット	16バイト(4ワード)
ライン3	20ビット	1ビット	1ビット	1ビット	16バイト(4ワード)
ライン4	20ビット	1ビット	1ビット	1ビット	16バイト(4ワード)
：	：	：	：	：	：
：	：	：	：	：	：
：	：	：	：	：	：
：	：	：	：	：	：
：	：	：	：	：	：
：	：	：	：	：	：

【図５】

	ラインの状態			意　味
	validフラグ	dirtyフラグ	stackフラグ	
（ａ）	0	0	0	tagが指すメモリの内容はラインの内容よりも新しい （現在このラインは開放されている）
（ｂ）	0	0	1	ラインはスタックに割り付けられていた tagが指すメモリの内容はラインの内容よりも新しい （現在このラインは開放されている）
（ｃ）	0	1	0	－
（ｄ）	0	1	1	－
（ｅ）	1	0	0	tagが指すメモリの内容とラインの内容は一致している
（ｆ）	1	0	1	ラインはスタックに割り付けられている tagが指すメモリの内容とラインの内容は一致している
（ｇ）	1	1	0	ラインの内容はtagが指すメモリの内容よりも新しい
（ｈ）	1	1	1	ラインはスタックに割り付けられている ラインの内容はtagが指すメモリの内容よりも新しい

【図６】

	ラインの状態			動　作	
	validフラグ	dirtyフラグ	stackフラグ	ランダムリード	ランダムライト
（ａ）	未割り当て			開放するラインの決定 決定したラインの開放 リフィル （ｂ）の動作	開放するラインの決定 決定したラインの開放 リフィル （ｂ）の動作
（ｂ）	1	0	0	データリード	データライト d=1
（ｃ）	1	0	1	データリード s=0	データライト d=1 s=0
（ｄ）	1	1	0	データリード	データライト
（ｅ）	1	1	1	データリード s=0	データライト s=0
（ｆ）	0	0	0	リフィル v=1 データリード	リフィル v=1 データライト d=1
（ｇ）	0	0	1	リフィル v=1 データリード s=0	リフィル v=1 データライト d=1 s=0

【図７】

	ラインの状態			動　作	
	validフラグ	dirtyフラグ	stackフラグ	SPが上方に移動 (SP−tag ≧ 4ワード)	ラインの開放
(a)	未割り当て			−	
(b)	1	0	0		v=0
(c)	1	0	1		v=0 s=0
(d)	1	1	0		ライトバック v=0 d=0
(e)	1	1	1	ライトバック d=0	ライトバック v=0 d=0 s=0
(f)	0	0	0		
(g)	0	0	1		s=0

【図８】

	ラインの状態			動　作	
	validフラグ	dirtyフラグ	stackフラグ	SP相対リード	SP相対ライト
（ａ）	未割り当て			開放するラインの決定 決定したラインの開放 リフィル （ｂ）の動作	開放するラインの決定 決定したラインの開放 リフィル （ｂ）の動作
（ｂ）	1	0	0	データリード	データライト　d＝1
（ｃ）	1	0	1	データリード	データライト　d＝1
（ｄ）	1	1	0	データリード	データライト
（ｅ）	1	1	1	データリード	データライト
（ｆ）	0	0	0	リフィル　v＝1 データリード	リフィル　v＝1 データライト　d＝1
（ｇ）	0	0	1	リフィル　v＝1 データリード	リフィル　v＝1 データライト　d＝1

【図９】

	ラインの状態			動　作	
	validフラグ	dirtyフラグ	stackフラグ	最上位アドレスにpush	最上位アドレス以外にpush
（ａ）	未割り当て			開放するラインの決定 決定したラインの開放 リフィル　ｖ＝１　ｄ＝０ ｓ＝０　（ｂ）の動作	開放するラインの決定 決定したラインの開放 リフィル　ｖ＝１　ｄ＝０ ｓ＝０　（ｂ）の動作
（ｂ）	１	０	０	データライト　ｄ＝１　ｓ＝１	データライト　ｄ＝１
（ｃ）	１	０	１	データライト　ｄ＝１	データライト　ｄ＝１
（ｄ）	１	１	０	データライト　ｓ＝１	データライト
（ｅ）	１	１	１	データライト	データライト
（ｆ）	０	０	０	リフィル　ｖ＝１ データライト　ｄ＝１　ｓ＝１	リフィル　ｖ＝１ データライト　ｄ＝１
（ｇ）	０	０	１	リフィル　ｖ＝１ データライト　ｄ＝１	リフィル　ｖ＝１ データライト　ｄ＝１

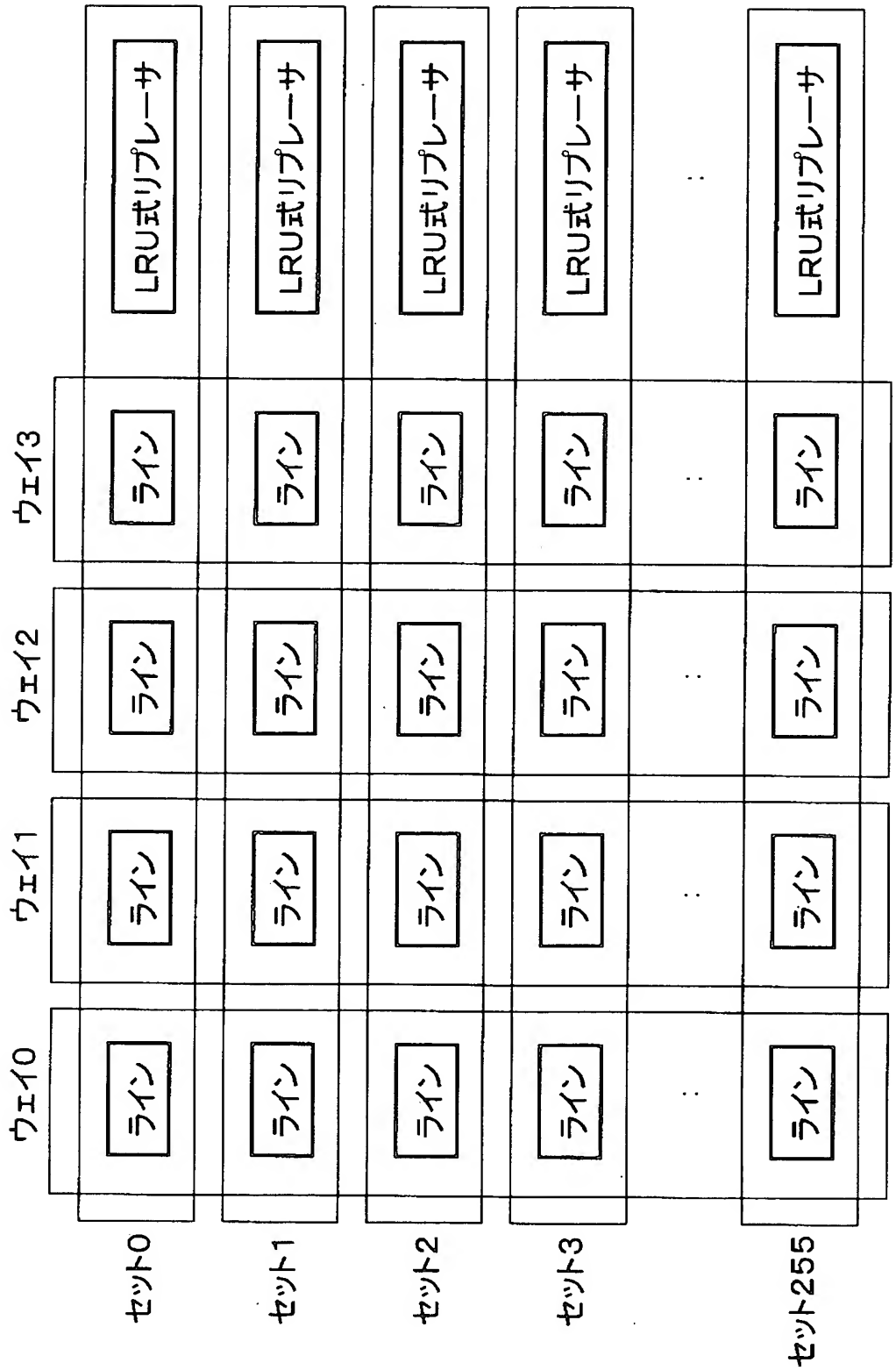
【図１０】

	ラインの状態			動　作	
	validフラグ	dirtyフラグ	stackフラグ	SPが下方に遠ざかった (tag－SP ≧ 12ワード)	SPが下方から近付いた (tag－SP ≦ 8ワード)
（a）	未割り当て			開放するラインの決定 決定したラインの開放 リフィル （g）の動作	
（b）	1	0	0		
（c）	1	0	1		
（d）	1	1	0		
（e）	1	1	1	ライトバック　d=0	
（f）	0	0	0		
（g）	0	0	1	リフィル　v=1	

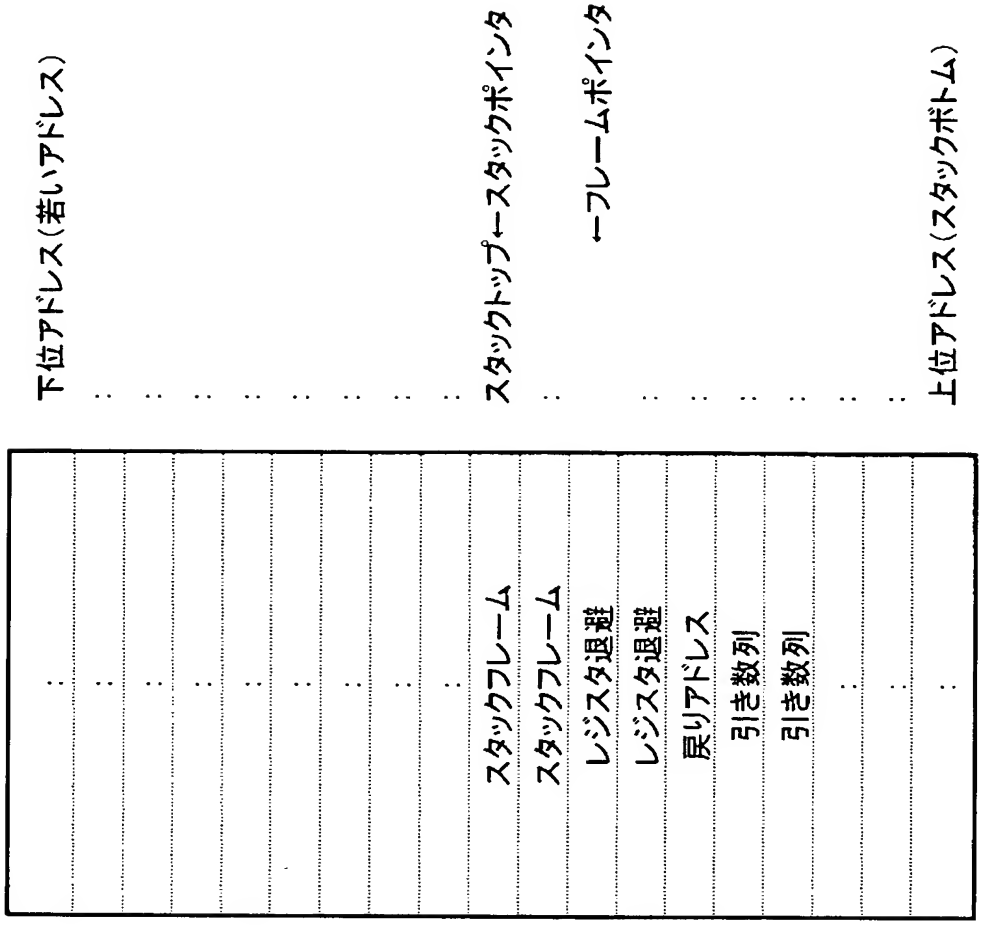
【図１１】

	ラインの状態			動　作	
	validフラグ	dirtyフラグ	stackフラグ	最上位アドレスからpop	最上位アドレス以外からpop
（a）	未割り当て			開放するラインの決定 決定したラインの開放 リフィル v=1 d=0 s=1 データリード v=0	開放するラインの決定 決定したラインの開放 リフィル v=1 d=0 データリード
（b）	1	0	0	データリード v=0 s=1	データリード
（c）	1	0	1	データリード v=0	データリード
（d）	1	1	0	データリード v=0 d=0 s=1	データリード
（e）	1	1	1	データリード v=0 d=0	データリード
（f）	0	0	0	リフィル v=1 データリード v=0 s=1	リフィル v=1 データリード
（g）	0	0	1	リフィル v=1 データリード v=0	リフィル v=1 データリード

【図１２】


ウェイ0
ウェイ1
ウェイ2
ウェイ3
セット0
セット1
セット2
セット3
セット255
ライン
LRU式リプレーサ

【図１３】



C言語の
スタック構造を示す
図

【図１４】

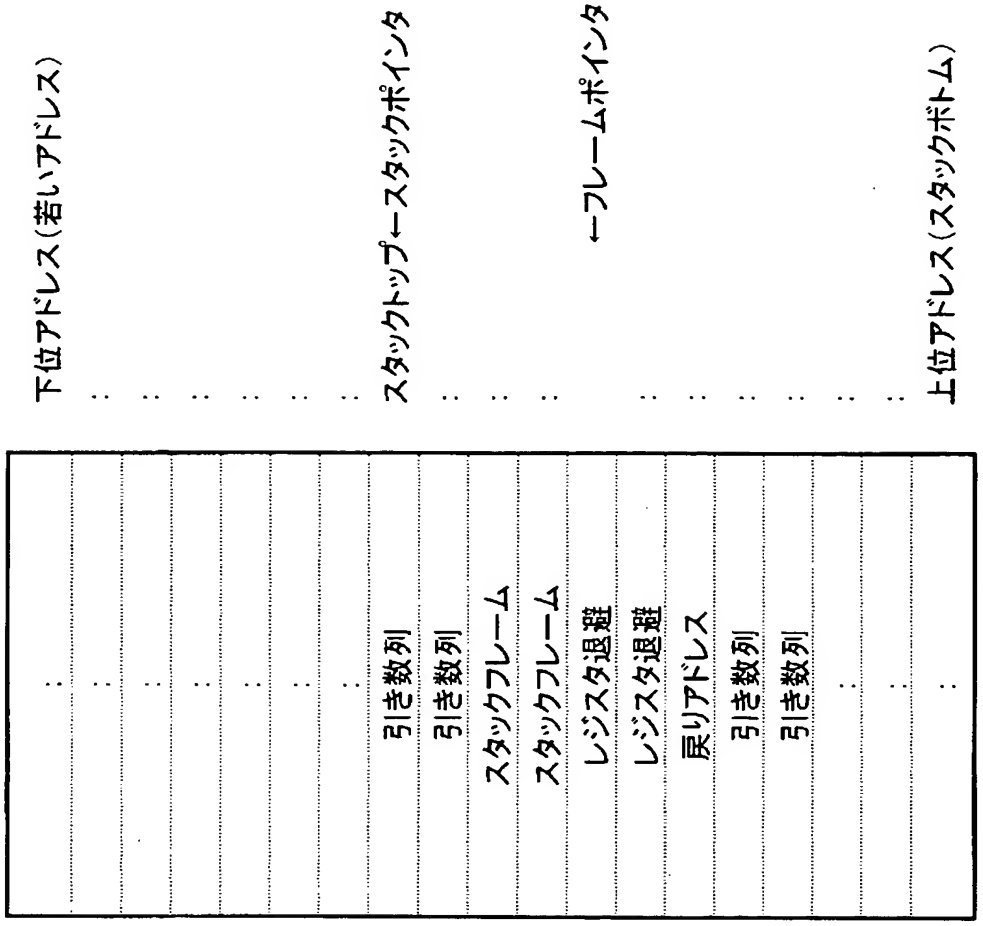
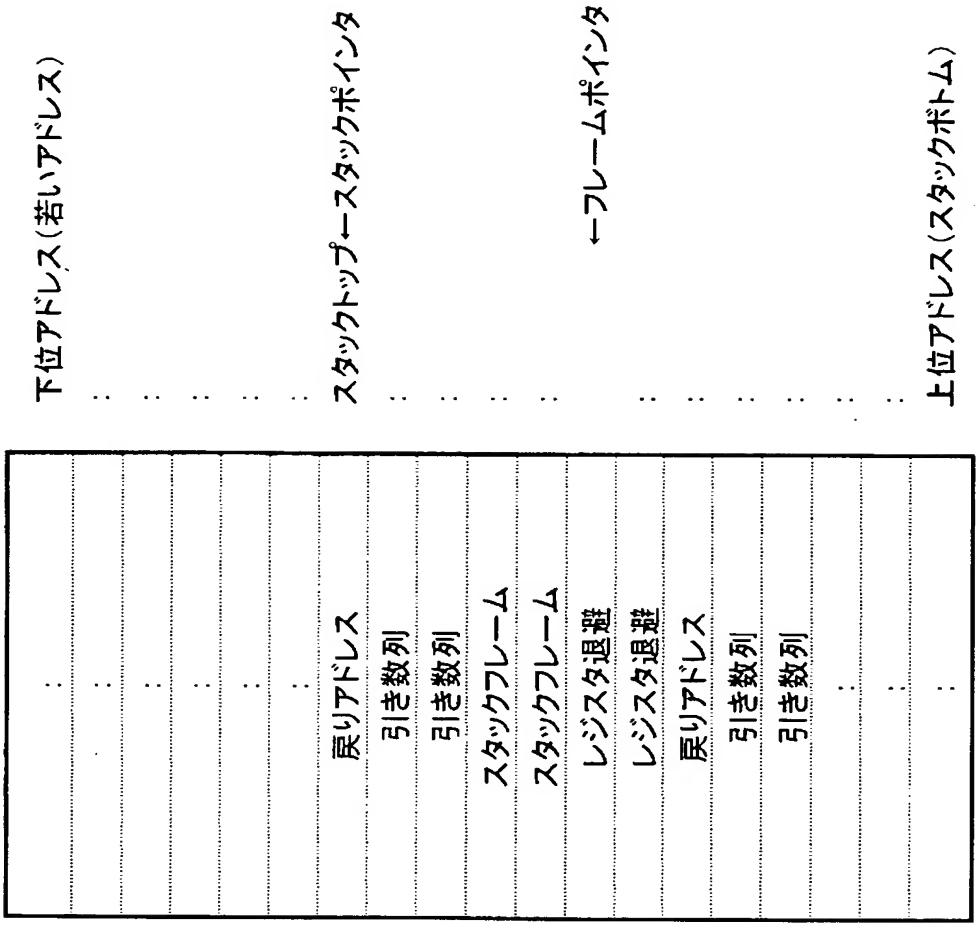


図　引き数生成直後のスタック構造を示す

【図１５】



関数呼び出し直後のスタック構造を示す図

【図１６】

レジスタ退避直後のスタック構造を示す図

【図１７】

スタックフレーム生成直後のスタック構造を示す図

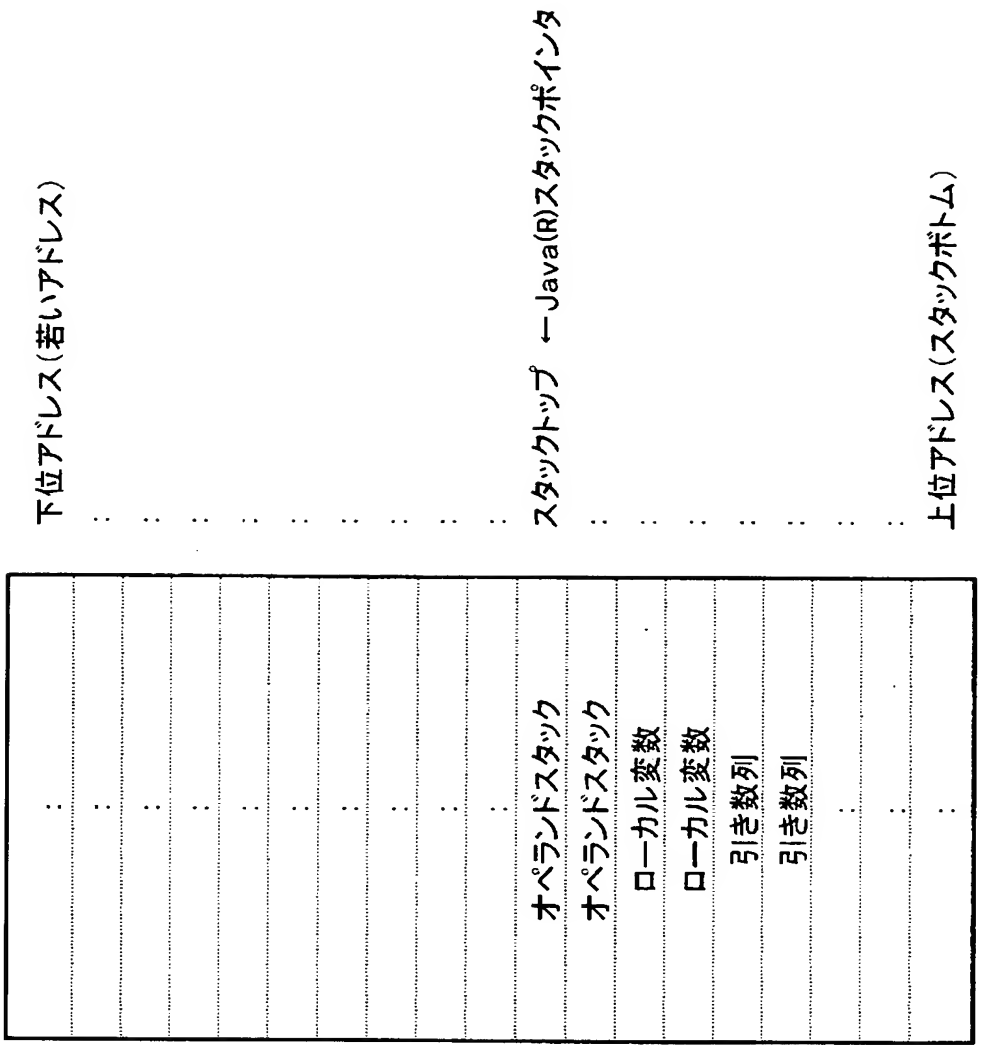
【図１８】

スタックフレーム生成
直後の
キャッシュラインの状態
を示す図
下位アドレス(若いアドレス)
未割り当て
未割り当て
v=1 d=1 s=1
v=1 d=1 s=1
v=1 d=1 s=1
v=1 d=1 s=1
v=1 d=1 s=1
スタックフレーム
スタックフレーム
レジスタ退避
レジスタ退避
戻りアドレス
引き数列
引き数列
スタックフレーム
スタックフレーム
レジスタ退避
レジスタ退避
戻りアドレス
引き数列
引き数列
上位アドレス(スタックボトム)

【図１９】

ランダムアクセス
後の
キャッシュラインの状態
を示す図

【図２０】



Java(R)スタック構造
を示す図

【図２１】

引き数生成直後の
Java(R)スタック構造
を示す図

【図２２】

ローカル変数領域
生成直後の
Java(R)スタック構造
を示す図

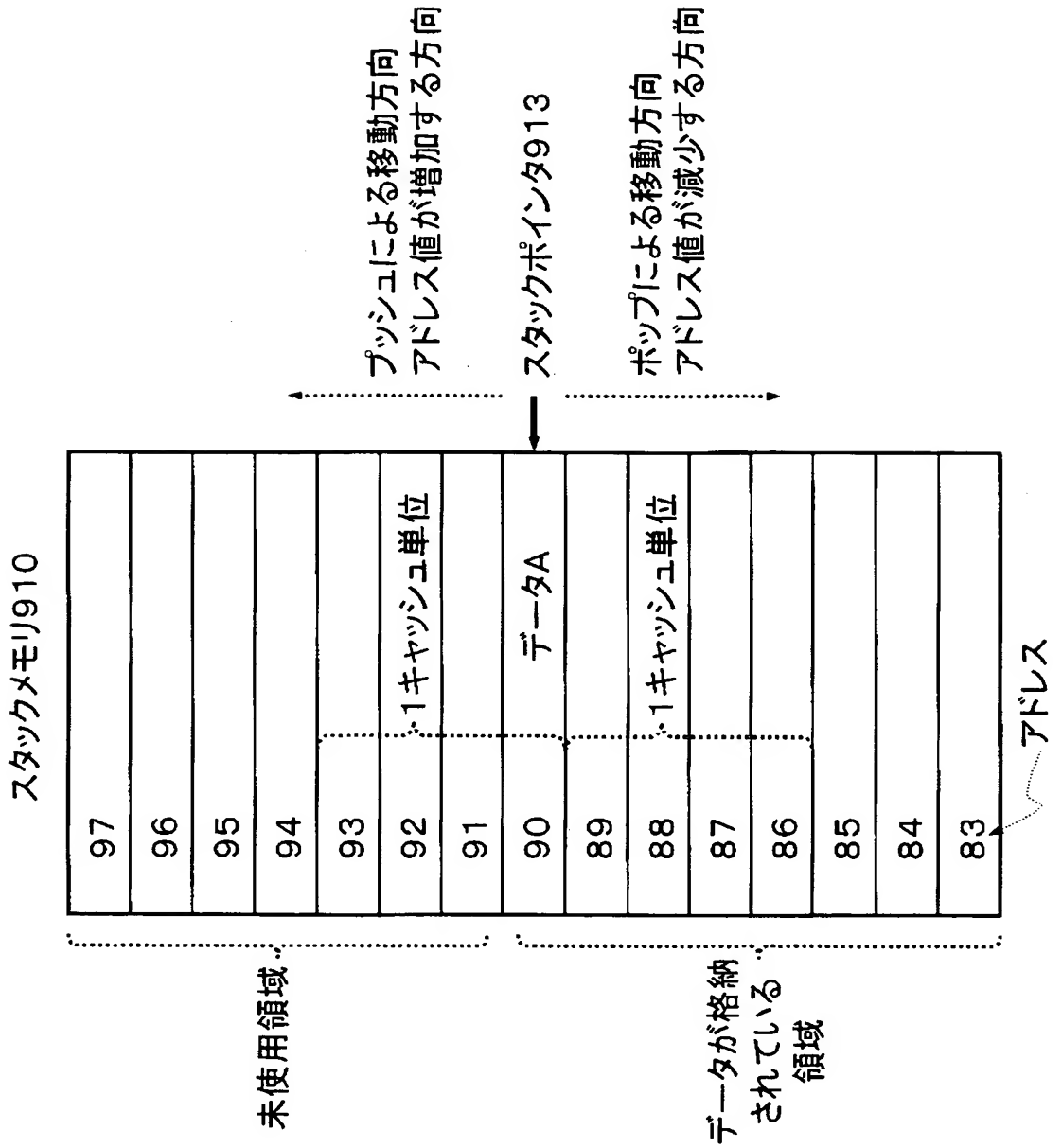
【図２３】

関数処理中の
Java(R)スタック構造
を示す図

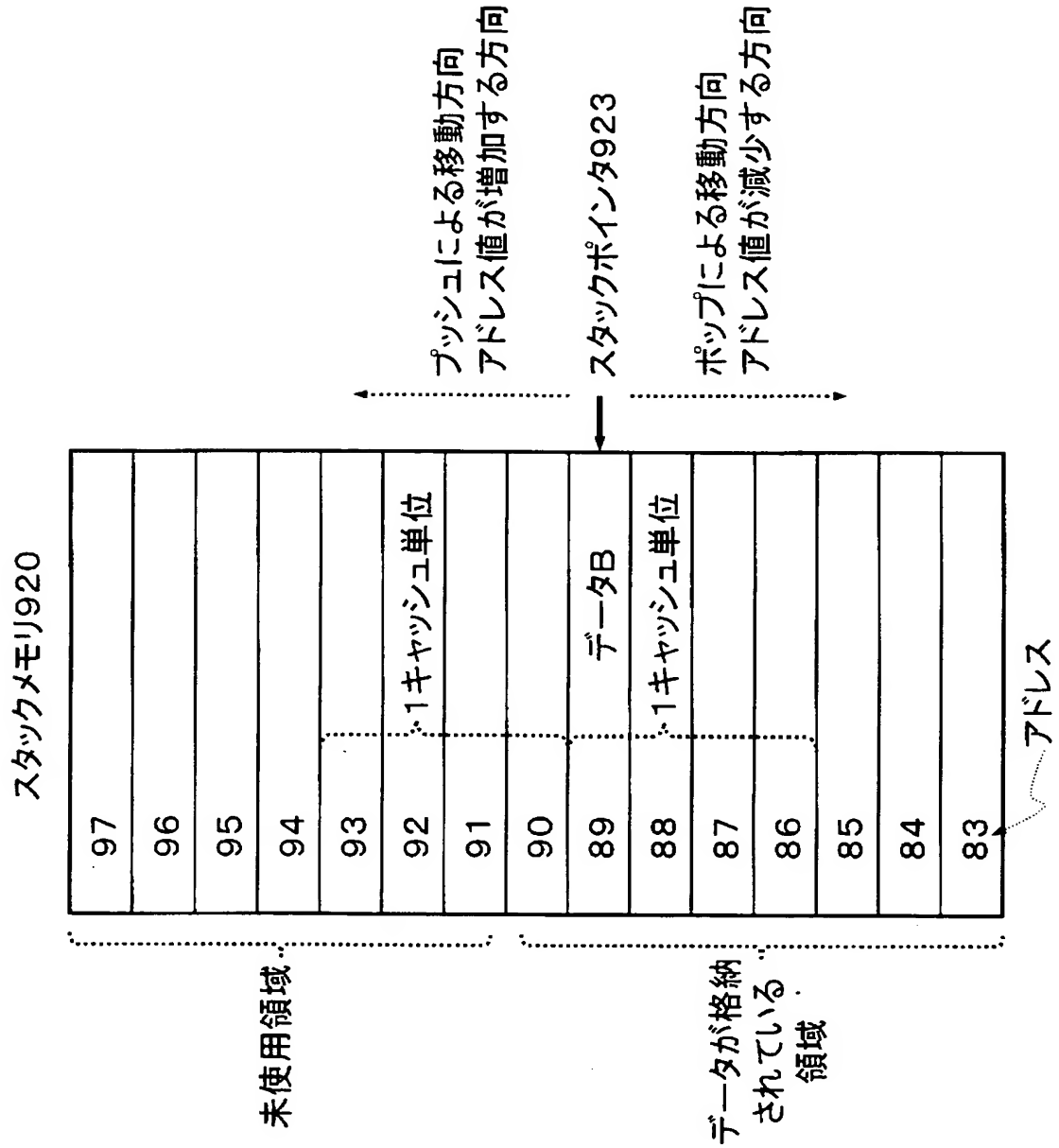
【図２４】

関数処理中の
キャッシュラインの
状態を示す図

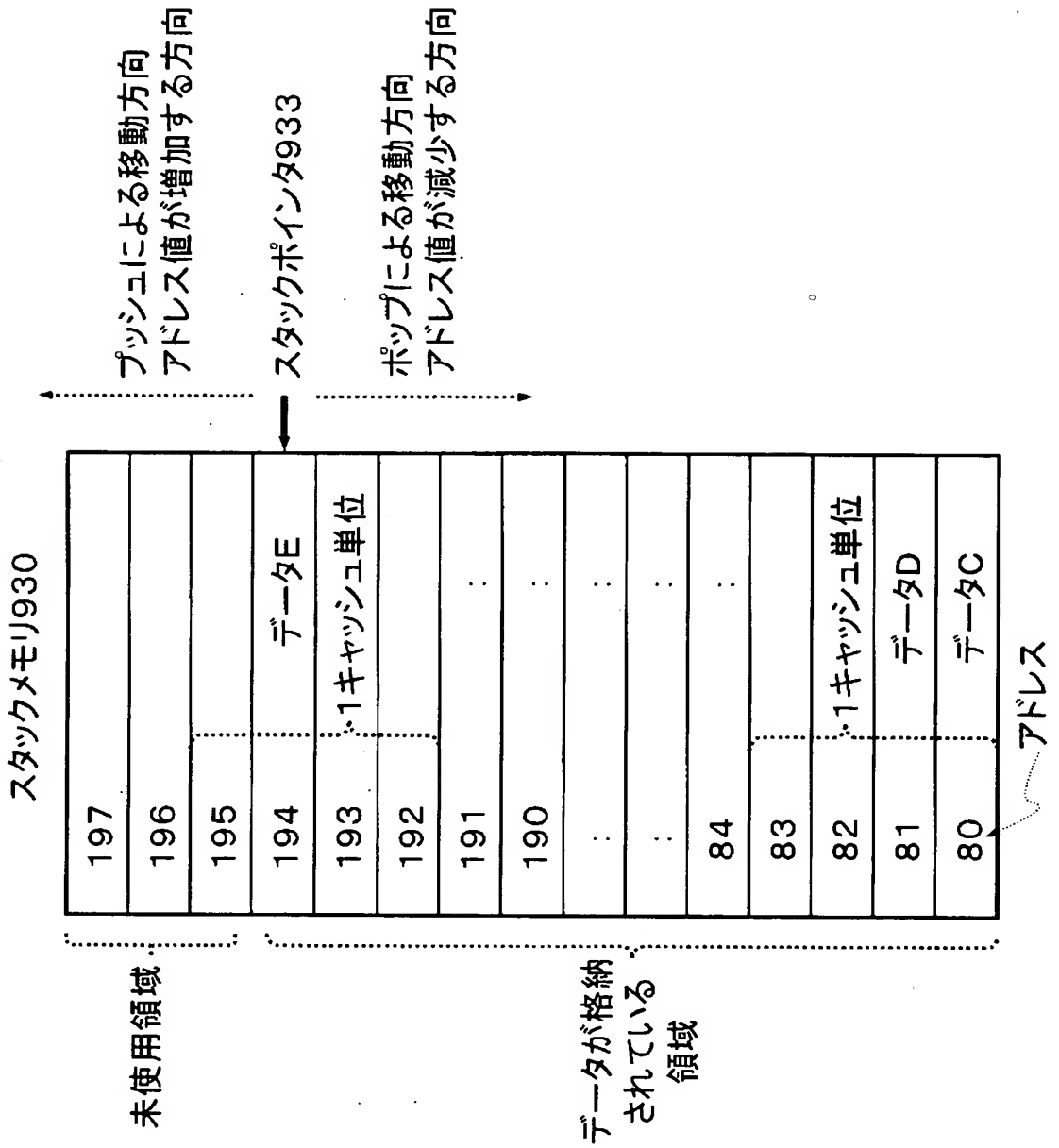
【図２５】


プッシュによる移動方向
アドレス値が増加する方向
スタックポインタ913
ポップによる移動方向
アドレス値が減少する方向
スタックメモリ910
97
96
95
94
93
92
91
90
89
88
87
86
85
84
83
1キャッシュ単位
データA
1キャッシュ単位
アドレス
未使用領域
データが格納されている領域

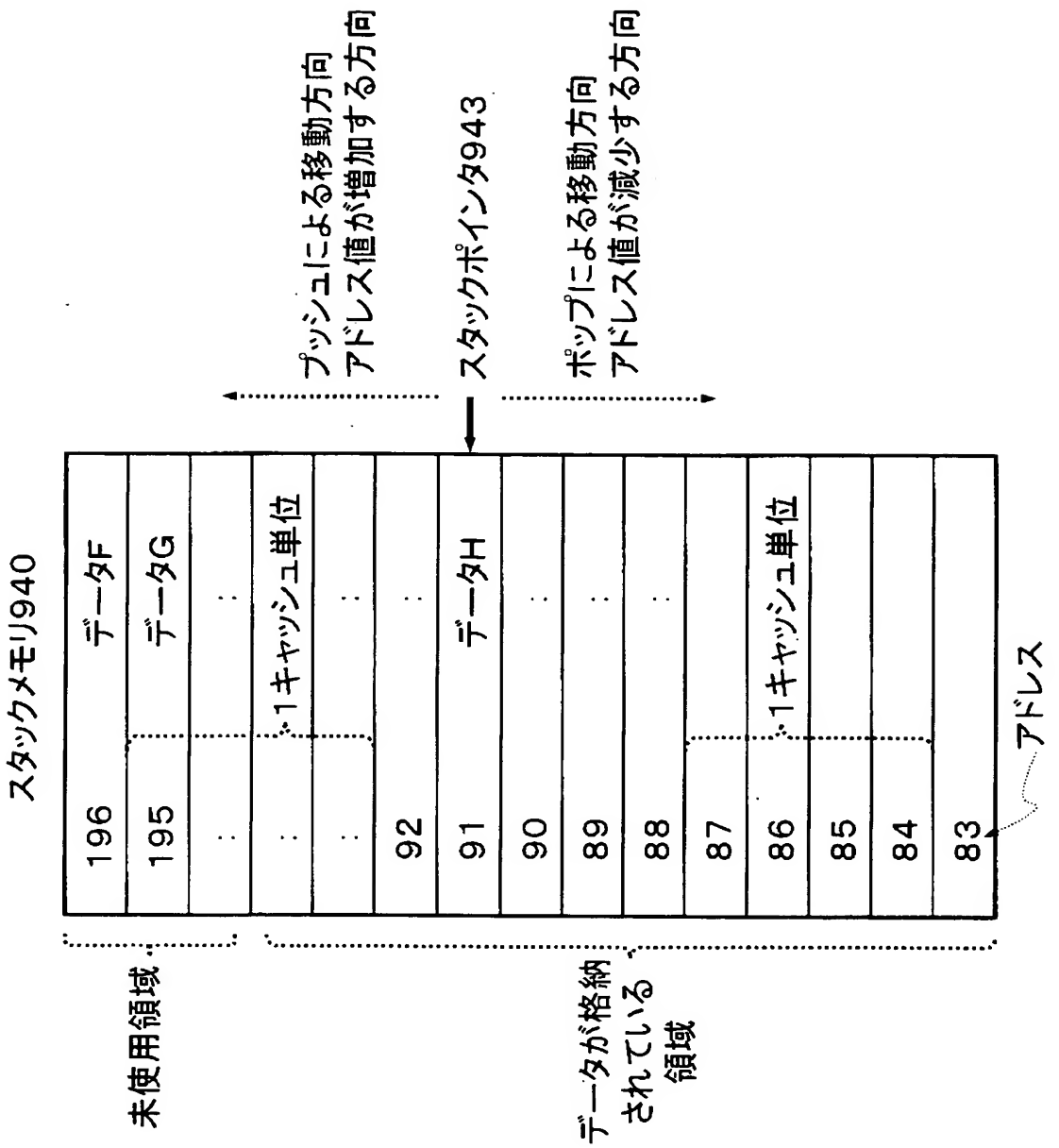
【図２６】


プッシュによる移動方向
アドレス値が増加する方向
スタックポインタ923
ポップによる移動方向
アドレス値が減少する方向
スタックメモリ920
97
96
95
94
93
92
91
90
89
88
87
86
85
84
83
1キャッシュ単位
データB
1キャッシュ単位
アドレス
未使用領域
データが格納されている領域

【図２７】


プッシュによる移動方向
アドレス値が増加する方向
スタックポインタ933
ポップによる移動方向
アドレス値が減少する方向
スタックメモリ930
197
196
195
194
193
192
191
190
84
83
82
81
80
データE
1キャッシュ単位
データD
データC
アドレス
未使用領域
データが格納
されている
領域

【図２８】


スタックメモリ940
196
195
92
91
90
89
88
87
86
85
84
83
データF
データG
データH
1キャッシュ単位
1キャッシュ単位
アドレス
プッシュによる移動方向
アドレス値が増加する方向
スタックポインタ943
ポップによる移動方向
アドレス値が減少する方向
未使用領域
データが格納
されている
領域

【書類名】　要約書

【要約】

【課題】　スタックに対するメモリアクセスの特性を考慮することで、キャッシュ制御の効率化を行う。

【解決手段】　連続したメモリ領域に対する連続書き込み時にキャッシュミスが発生した時は次に書き込みが行われるメモリ領域に対してリフィルを行わず、連続したメモリ領域に対する連続読み込み時にキャッシュミスが発生した時は既に読み込みが行われたメモリ領域に対してライトバックを行わず、スタックトップから一定以上離れたメモリ領域に対してはバスの空き時間を使って予めライトバックを行い、スタックトップから一定以上近いメモリ領域に対してはバスの空き時間を使って予めリフィルを行う、データキャッシュメモリ制御装置。

【選択図】　図１

特願２００３－０７８０２６

出　願　人　履　歴　情　報

識別番号　　［０００００５８２１］

１．変更年月日　　１９９０年　８月２８日
［変更理由］　　新規登録
住　所　　大阪府門真市大字門真１００６番地
氏　名　　松下電器産業株式会社